Oracle® Fusion Middleware

Administering Security for Oracle WebLogic Server

14c (14.1.2.0.0)
F62227-03
April 2025

Oracle Fusion Middleware Administering Security for Oracle WebLogic Server, 14c (14.1.2.0.0)

F62227-03

Primary Author: Oracle Corporation

Contents

12 Configuring LDAP Authentication Providers

13 Configuring RDBMS Authentication Providers

14 Configuring the SAML Authentication Provider

15 Configuring the Password Validation Provider

16 Configuring Identity Assertion Providers

23 Configuring SAML 2.0 Services

27 Managing the Embedded LDAP Server

Part VI Configuring SSL

28 Overview of Configuring SSL in WebLogic Server

29 Configuring Keystores

38 Using the JSSE-Based SSL Implementation

39 X.509 Certificate Revocation Checking

Preface

This document explains how to configure WebLogic Server security, including settings for security realms, providers, identity and trust, SSL, and compatibility security.

Audience

This document is intended for the following audiences:

- Application Architects—Architects who, in addition to setting security goals and designing the overall security architecture for their organizations, evaluate WebLogic Server security features and determine how to best implement them. Application Architects have in-depth knowledge of Java programming, Java security, and network security, as well as knowledge of security systems and leading-edge, security technologies and tools.
- Security Developers—Developers who define the system architecture and infrastructure for security products that integrate with WebLogic Server and who develop custom security providers for use with WebLogic Server. They work with Application Architects to ensure that the security architecture is implemented according to design and that no security holes are introduced, and work with Server Administrators to ensure that security is properly configured. Security Developers have a solid understanding of security concepts, including authentication, authorization, auditing (AAA), in-depth knowledge of Java (including Java Management eXtensions (JMX)), and working knowledge of WebLogic Server and security provider functionality.
- Application Developers—Java programmers who focus on developing client applications, adding security to Web applications and Enterprise JavaBeans (EJBs), and working with other engineering, quality assurance (QA), and database teams to implement security features. Application Developers have in-depth/working knowledge of Java (including Java EE components such as servlets/JSPs and JSEE) and Java security.
- Server Administrators—Administrators work closely with Application Architects to design a security scheme for the server and the applications running on the server; to identify potential security risks; and to propose configurations that prevent security problems. Related responsibilities may include maintaining critical production systems; configuring and managing security realms, implementing authentication and authorization schemes for server and application resources; upgrading security features; and maintaining security provider databases. Server Administrators have in-depth knowledge of the Java security architecture, including Web services, Web application and EJB security, Public Key security, SSL, and Security Assertion Markup Language (SAML).
- Application Administrators—Administrators who work with Server Administrators to implement and maintain security configurations and authentication and authorization schemes, and to set up and maintain access to deployed application resources in defined security realms. Application Administrators have general knowledge of security concepts and the Java Security architecture. They understand Java, XML, deployment descriptors, and can identify security events in server and audit logs.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Information

The following Oracle Fusion Middleware documents contain information that is relevant to the WebLogic Security Service:

- *Understanding Security for Oracle WebLogic Server*—Summarizes the features of the WebLogic Security Service, including an overview of its architecture and capabilities. It is the starting point for understanding WebLogic security.
- *Developing Security Providers for Oracle WebLogic Server*—Provides security vendors and application developers with the information needed to develop custom security providers that can be used with WebLogic Server.
- *Securing a Production Environment for Oracle WebLogic Server*—Highlights essential security hardening and lockdown measures for you to consider before you deploy WebLogic Server in a production environment.
- *Securing Resources Using Roles and Policies for Oracle WebLogic Server*—Introduces the various types of WebLogic resources, and provides information about how to secure these resources using WebLogic Server. This document focuses primarily on securing URL (Web) and Enterprise JavaBean (EJB) resources.
- *Developing Applications with the WebLogic Security Service* —Describes how to develop secure Web applications. in
- *Securing WebLogic Web Services for Oracle WebLogic Server*—Describes how to develop and configure secure Web services.
- *Oracle WebLogic Remote Console Online Help*—Many security configuration tasks can be performed using the WebLogic Remote Console. The online help describes configuration procedures and provides a reference for configurable attributes.
- *Upgrading Oracle WebLogic Server*—Provides procedures and other information you need to upgrade from earlier versions of WebLogic Server to this release. It also provides

information about moving applications from an earlier version of WebLogic Server to this release.

- *Java API Reference for Oracle WebLogic Server*—Provides reference documentation for the WebLogic security packages that are provided with and supported by this release of WebLogic Server.

Security Examples in the WebLogic Server Distribution

WebLogic Server optionally installs API code examples in *EXAMPLES_HOME*/examples/src/examples/security, where *EXAMPLES_HOME* represents the directory in which the WebLogic Server code examples are configured. By default, this location is *ORACLE_HOME*/wlserver/samples/server. For more information about the WebLogic Server code examples, see Sample Applications and Code Examples in *Understanding Oracle WebLogic Server*.

The following examples illustrate WebLogic security features:

- Java Authentication and Authorization Service
- SAML 2.0 For Web SSO Scenario
- Outbound and Two-way SSL

The WebLogic Server installation also includes an example demonstrating the use of the built-in database identity store functionality provided by the Java EE Security API (JSR 375). This example is located in the *EXAMPLES_HOME*/examples/src/examples/javaee8/security directory.

New and Changed WebLogic Server Features

For a comprehensive listing of the new WebLogic Server features introduced in this release, see *What's New in Oracle WebLogic Server*.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Part I

Overview of WebLogic Server Security Administration

Before you begin administering Oracle WebLogic Server security, you need to understand some basic concepts about WebLogic Server security management, the set of security standards supported by WebLogic Server, and the tasks involved in securing a WebLogic domain.

This part contains the following chapters:

1

Security Management Concepts

The task of managing Oracle WebLogic Server security focuses primarily on creating and configuring one or more security realms. In each security realm, you configure a set of security providers, create security policies for the WebLogic resources that need to be protected, select configuration options for protecting user accounts, and configure identity and trust.

This chapter includes the following sections:

For a broader overview of security management concepts, see *Understanding Security for Oracle WebLogic Server*.

Security Realms in WebLogic Server

The security service in WebLogic Server simplifies the configuration and management of security while offering robust capabilities for securing your WebLogic Server deployment. Security realms act as a scoping mechanism. Each security realm consists of a set of configured security providers, users, groups, security roles, and security policies. You can configure and activate multiple security realms in a domain; however, only one can be the default administrative realm.
WebLogic Server provides a default security realm, myrealm, which has the WebLogic Adjudication, Authentication, Identity Assertion, Authorization, Role Mapping, and Credential Mapping providers configured by default.

You can customize authentication and authorization functions by configuring a new security realm to provide the security services you want and then set the new security realm as the default security realm.

For information about the default security configuration in WebLogic Server, see The Default Security Configuration in WebLogic Server.

For information about configuring a security realm and setting it as the default security realm, see Customizing the Default Security Configuration.

Security Providers

Security providers are modular components that handle specific aspects of security, such as authentication and authorization. Although applications can leverage the services offered by the default WebLogic security providers, the WebLogic Security Service's flexible infrastructure also allows security vendors to write their own custom security providers for use with WebLogic

Server. WebLogic security providers and custom security providers can be mixed and matched to create unique security solutions, allowing organizations to take advantage of new technology advances in some areas while retaining proven methods in others. The WebLogic Remote Console allows you to administer and manage all your security providers through one unified management interface.
The WebLogic Security Service supports the following types of security providers:

- **Authentication**—Authentication is the process whereby the identity of users or system processes are proved or verified. Authentication also involves remembering, transporting, and making identity information available to various components of a system when that information is needed. Authentication providers supported by the WebLogic Security Service supply the following types of authentication:
  - Username and password authentication
  - Certificate-based authentication directly with WebLogic Server
  - HTTP certificate-based authentication proxied through an external Web server
- **Identity Assertion**—An Authentication provider that performs perimeter authentication—a special type of authentication using tokens—is called an Identity Assertion provider. Identity assertion involves establishing a client's identity through the use of client-supplied tokens that may exist outside of the request. Thus, the function of an Identity Assertion provider is to validate and map a token to a username. Once this mapping is complete, an Authentication provider's LoginModule can be used to convert the username to a principal (an authenticated user, group, or system process).
- **Authorization**—Authorization is the process whereby the interactions between users and WebLogic resources are limited to ensure integrity, confidentiality, and availability. In other words, once a user's identity has been established by an authentication provider, authorization is responsible for determining whether access to WebLogic resources should be permitted for that user. An Authorization provider supplies these services.
- **Role Mapping**—You can assign one or more roles to multiple users and then specify access rights for users who hold particular roles. A Role Mapping provider obtains a computed set of roles granted to a requestor for a given resource. Role Mapping providers supply Authorization providers with this information so that the Authorization provider can answer the "is access allowed?" question for WebLogic resources that use role-based security (for example, Web applications and Enterprise JavaBeans (EJBs)).
- **Adjudication**—When multiple Authorization providers are configured in a security realm, each may return a different answer to the "is access allowed" question for a given resource. Determining what to do if multiple Authorization providers do not agree is the primary function of an Adjudication provider. Adjudication providers resolve authorization conflicts by weighing each Authorization provider's answer and returning a final access decision.
- **Credential Mapping**—A credential map is a mapping of credentials used by WebLogic Server to credentials used in a legacy or remote system, which tell WebLogic Server how to connect to a given resource in that system. In other words, credential maps allow WebLogic Server to log into a remote system on behalf of a subject that has already been authenticated. Credential Mapping providers map credentials in this way.
- **Keystore**—A keystore is a mechanism for creating and managing password-protected stores of private keys and certificates for trusted certificate authorities. The keystore is available to applications that may need it for authentication or signing purposes. In the WebLogic Server security architecture, the WebLogic Keystore provider is used to access keystores.

> **Note:**
>
> The WebLogic Server Keystore provider is removed and is only supported for backward compatibility. Use JDK keystore instead. For more information about configuring keystores, see Creating a Keystore.

- **Certificate Lookup and Validation (CLV)**—X.509 certificates need to be located and validated for purposes of identity and trust. CLV providers receive certificates, certificate chains, or certificate references, complete the certificate path (if necessary), and validate all the certificates in the path. There are two types of CLV providers:
  - A CertPath Builder looks up and optionally completes the certificate path and validates the certificates.
  - A CertPath Validator looks up and optionally completes the certificate path, validates the certificates, and performs extra validation (for example, revocation checking).
- **Certificate Registry**—A certificate registry is a mechanism for adding certificate revocation checking to a security realm. The registry stores a list of valid certificates. Only registered certificates are valid. A certificate is revoked by removing it from the certificate registry. The registry is stored in the embedded LDAP server. The Certificate Registry is both a CertPath Builder and a CertPath Validator.
- **Auditing**—Auditing is the process whereby information about security requests and the outcome of those security requests is collected, stored, and distributed for the purpose of non-repudiation. In other words, auditing provides an electronic trail of computer activity. An Auditing provider supplies these services.

For information about the functionality provided by the WebLogic security providers, see About Configuring WebLogic Security Providers and About Configuring the Authentication Providers in WebLogic Server.

For information about the default security configuration, see The Default Security Configuration in WebLogic Server.

For information about writing custom security providers, see *Developing Security Providers for Oracle WebLogic Server*.

Security Policies and WebLogic Resources

WebLogic Server uses security policies to protect WebLogic resources. Security policies answer the question "who has access" to a WebLogic resource. A security policy is created when you define an association between a WebLogic resource and a user, group, or security role. You can also optionally associate a time constraint with a security policy. A WebLogic resource has no protection until you assign it a security policy.
Creating security policies is a multi-step process with many options. To fully understand this process, read *Securing Resources Using Roles and Policies for Oracle WebLogic Server*. That document should be used in conjunction with Securing WebLogic Security to ensure security is completely configured for a WebLogic Server deployment.

This section includes the following topics:

- WebLogic Resources
- Deployment Descriptors and WebLogic Remote Console

WebLogic Resources

A WebLogic resource is a structured object used to represent an underlying WebLogic Server entity, which can be protected from unauthorized access. WebLogic Server defines the following resources:

- Administrative resources such as the WebLogic Remote Console and WebLogic Scripting Tool.
- Application resources that represent Enterprise applications. This type of resource includes individual EAR (Enterprise Application aRchive) files and individual components, such as EJB JAR files contained within the EAR.
- Component Object Model (COM) resources that are designed as program component objects according to Microsoft's framework. This type of resource includes COM components accessed through the Oracle bidirectional COM-Java (jCOM) bridging tool.
- Enterprise Information System (EIS) resources that are designed as resource adapters, which allow the integration of Java applications with existing enterprise information systems. These resource adapters are also known as connectors.
- Enterprise JavaBean (EJB) resources including EJB JAR files, individual EJBs within an EJB JAR, and individual methods on an EJB.
- Java DataBase Connectivity (JDBC) resources including groups of connection pools, individual connection pools, and multipools.
- Java Naming and Directory Interface (JNDI) resources.
- Java Messaging Service (JMS) resources.
- Server resources related to WebLogic Server instances, or servers. This type of resource includes operations that start, shut down, lock, or unlock servers.
- URL resources related to Web applications. This type of resource can be a Web Application aRchive (WAR) file or individual components of a Web application (such as servlets and JSPs).

> **Note:**
>
> Web resources are deprecated. Use the URL resource instead.

- Web services resources related to services that can be shared by and used as components of distributed, Web-based applications. This type of resource can be an entire Web service or individual components of a Web service (such as a stateless session EJB, particular methods in that EJB, the Web application that contains the `web-services.xml` file, and so on).
- Remote resources.

Deployment Descriptors and WebLogic Remote Console

The WebLogic Security Service can use information defined in deployment descriptors to grant security roles and define security policies for Web applications and EJBs.
WebLogic Server offers a choice of models for configuring security roles and policies. Under the standard Java Enterprise Edition model, you define role mappings and policies in the Web application or EJB deployment descriptors. The WebLogic Security Service can use information defined in deployment descriptors to grant security roles and define security

policies for Web applications and EJBs. When WebLogic Server is booted for the first time, security role and security policy information stored in `web.xml`, `weblogic.xml`, `ejb-jar.xml`, or `weblogic-ejb-jar.xml` deployment descriptors is loaded into the Authorization and Role Mapping providers configured in the default security realm. You can then view the role and policy information from WebLogic Remote Console. (Optionally, you may configure the security realm to use a different security model that allows you to make changes to that information using WebLogic Remote Console as well.)

To use information in deployment descriptors, at least one Authorization and Role Mapping provider in the security realm must implement the `DeployableAuthorizationProvider` and `DeployableRoleProvider` Security Service Provider Interface (SSPI). This SSPI allows the providers to store (rather than retrieve) information from deployment descriptors. By default, the WebLogic Authorization and Role Mapping providers implement this SSPI.

If you change security role and security policy in deployment descriptors through WebLogic Remote Console and want to continue to modify this information through WebLogic Remote Console, you can set configuration options on the security realm to ensure changes made through the Console are not overwritten by old information in the deployment descriptors when WebLogic Server is rebooted.

See Options for Securing Web Application and EJB Resources in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*.

The Default Security Configuration in WebLogic Server

To simplify the configuration and management of security, WebLogic Server provides a default security configuration. In the default security configuration, `myrealm` is set as the default security realm, and the WebLogic Adjudication, Authentication, Identity Assertion, XACML Authorization, Credential Mapping, XACML Role Mapping, and CertPath providers are defined as the security providers in that realm. WebLogic Server's embedded LDAP server is used as the data store for these default security providers. To use the default security configuration, you need to define users, groups, and security roles for the security realm, and create security policies to protect the WebLogic resources in the domain.

> **Note:**
>
> WebLogic Server includes the WebLogic Authorization provider, which is referred to in the WebLogic Remote Console and elsewhere as the DefaultAuthorizer, and the WebLogic Role Mapping provider, which is referred to in the WebLogic Remote Console and elsewhere as the DefaultRoleMapper. Beginning with WebLogic Server 9.1, these providers are no longer the default providers in newly-created security realms. Instead, the XACML Authorization provider and the XACML Role Mapping provider are the default providers.
>
> The DefaultAuthorizer and the DefaultRoleMapper providers are deprecated in WebLogic Server 14.1.1.0.0, and will be removed in a future release.

For a description of the functionality provided by the WebLogic Security providers, see *Understanding Security for Oracle WebLogic Server*. If the WebLogic security providers do not fully meet your security requirements, you can supplement or replace them. See *Developing Security Providers for Oracle WebLogic Server*.

If the default security configuration does not meet your requirements, you can create a new security realm with any combination of WebLogic and custom security providers and then set

the new security realm as the default security realm. See Customizing the Default Security Configuration.

Configuring WebLogic Security: Main Steps

Because WebLogic Server's security features are closely related, it is difficult to determine where to start when configuring security. In fact, configuring security for your WebLogic Server deployment may be an iterative process. Although more than one sequence of steps may work, Oracle recommends the following procedure:

1. If you plan to use WebLogic Server in a production environment, make sure you do the following:
   a. Secure the host environment prior to installing WebLogic Server, as explained in Performing a Secure Installation of WebLogic Server.
   b. When creating the WebLogic domain, configure the domain to run in production mode or secured production mode, as explained in Creating a WebLogic Domain for Production Use.
   c. Immediately after starting the domain for the first time, complete the tasks described in Securing the Domain After You Have Created It.
2. Determine whether or not to use the default security configuration by reading Why Customize the Default Security Configuration?
   - If you are using the default security configuration, begin at step 4.
   - If you are not using the default security configuration, begin at step 3.
3. Configure additional security providers (for example, configure an LDAP Authentication provider instead of using the WebLogic Authentication provider) or configure custom security providers in the default security realm. This step is optional. By default, WebLogic Server configures the WebLogic security providers in the default security realm (`myrealm`). For information about the circumstances that require you to customize the default security configuration, see Why Customize the Default Security Configuration? For information about creating custom security providers, see Overview of the Development Process in *Developing Security Providers for Oracle WebLogic Server*.

   > **Note:**
   >
   > You can also create a new security realm, configure security providers (either WebLogic or custom) in the security realm and set the new security realm as the default security realm. See Customizing the Default Security Configuration.

4. Optionally, configure the embedded LDAP server. WebLogic Server's embedded LDAP server is configured with default options. However, you may want to change those options to optimize the use of the embedded LDAP server in your environment. See Managing the Embedded LDAP Server.
5. Ensure that user accounts are properly secured. WebLogic Server provides a set of configuration options for protecting user accounts. By default, they are set for maximum security. However, during the development and deployment of WebLogic Server, you may need to weaken the restrictions on user accounts. Before moving to production, check that the options on user accounts are set for maximum protection. If you are creating a new security realm, you need to set the user lockout options. See How Passwords Are Protected in WebLogic Server and Protecting User Accounts.

6. Protect WebLogic resources with security policies. Creating security policies is a multi-step process with many options. To fully understand this process and to ensure security is completely configured for a WebLogic Server deployment, read *Securing Resources Using Roles and Policies for Oracle WebLogic Server*.
7. Configure identity and trust for WebLogic Server. (This step is optional but strongly recommended, especially for production environments.) See Configuring Keystores.
8. Enable SSL/TLS for WebLogic Server. (This step is also optional, but strongly recommended for all production environments.) See Setting Up SSL: Main Steps.
9. When you have moved to production, review and implement the additional security options described in *Securing a Production Environment for Oracle WebLogic Server*.

In addition, you can:

- Configure a connection filter. See Using Connection Filters.
- Enable interoperability between WebLogic domains. See Configuring Cross-Domain Security.

Methods of Configuring Security

You can configure security for WebLogic Server using many tools. Typically, this document describes procedures for configuring securing using WebLogic Remote Console, but they can generally be accomplished using other configuration tools as well.

Common configuration tools include Fusion Middleware Control, WebLogic Scripting Tool (WLST), REST APIs, and the Java Management Extensions (JMX).

For information about using . . .	See the following topics . . .
WLST	Managing Security Data (WLST Online) in *Understanding the WebLogic Scripting Tool*
Fusion Middleware Control	WebLogic Server Security in *Administering Oracle WebLogic Server with Fusion Middleware Control*
REST	Using the WLS RESTful Management Interface in *Administering Oracle WebLogic Server with RESTful Management Services*
JMX APIs	Choosing an MBean Server to Manage Security Realms in *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*
WebLogic Remote Console	Securing Domains in *Oracle WebLogic Remote Console Online Help*

When you manage security realms, you must use two different MBean servers depending on your task:

- To set the value of a security MBean attribute, you must use the Edit MBean Server.
- To add users, groups, roles, and policies, or to invoke other operations in a security provider MBean, you must use a Runtime MBean Server or the Domain Runtime MBean Server.

For example, the value of the `MinimumPasswordLength` attribute in `DefaultAuthenticatorMBean` is stored in the domain's configuration document. Because all modifications to this document are controlled by WebLogic Server, to change the value of this attribute you must use the Edit MBean Server and acquire a lock on the domain's configuration. The `createUser` operation in `DefaultAuthenticatorMBean` adds data to an LDAP server, which is not controlled by WebLogic Server. To prevent incompatible changes

between the `DefaultAuthenticatorMBean`'s configuration and the data that it uses in the LDAP server, you cannot invoke the createUser operation if you or other users are in the process of modifying the `MinimumPasswordLength` attribute. In addition, because changing this attribute requires you to restart WebLogic Server, you cannot invoke the `createUser` operation until you have restarted the server.

How Passwords Are Protected in WebLogic Server

For WebLogic domain user accounts that are stored in the embedded LDAP, passwords are encrypted using a one-way hash that cannot be decrypted. Passwords for user accounts that are stored in an external LDAP system or RDBMS are stored in, and managed by, that LDAP or RDBMS. For external stores, the algorithm may vary, but most LDAP servers use one-way hashes; RDBMS systems use either hashes or encryption depending how they are configured. It is important to protect passwords that are used to access resources in a WebLogic domain. In the past, user names and passwords were stored in clear text in a WebLogic security realm.

> **Note:**
>
> The web services password digest feature in the WebLogic Authentication provider does not use hashed passwords. Instead, reversible encryption is used so that password digests can be computed at runtime. (Password digest authentication is not supported for servlets and web application.) For information about the Enable Password Digests attribute, see DefaultAuthenticatorMBean.PasswordDigestEnabled in *MBean Reference for Oracle WebLogic Server*.

The `SerializedSystemIni.dat` file contains the primary encryption key for the domain. It is associated with a specific WebLogic domain so it cannot be moved from domain to domain.

Sensitive configuration data, including such items as JDBC passwords, is encrypted with the primary encryption key. This encrypted data is kept in `config.xml`, or in the security metadata/ policy store in the embedded LDAP. (RDBMS is used if configured.)

If the `SerializedSystemIni.dat` file is destroyed or corrupted, it cannot be recovered. You will need to delete the existing WebLogic domain and create a new one. Therefore, you should take the following precautions:

- Make a backup copy of the `SerializedSystemIni.dat` file and put it in a safe location.
- Set permissions on the `SerializedSystemIni.dat` file such that the system administrator of a WebLogic Server deployment has write and read privileges and no other users have any privileges.

2

WebLogic Server Security Standards

The Oracle WebLogic Server WebLogic Security Service is built upon and supports standard Java EE security technologies such as the Java Authentication and Authorization Service (JAAS), Java Secure Sockets Extensions (JSSE), Java Cryptography Extensions (JCE), Java Authentication Service Provider Interface for Containers (JASPIC), Java Authorization Contract for Containers (JACC), the Java EE Security API (JSR 375), and more.

This chapter includes the following topics:

Supported Security Standards

WebLogic Server supports several Java EE security standards such as JAAS, JASPIC, JACC, JCE, the Java EE Security API (JSR 375), and more.

The complete set of supported security standards are provided in Table 2-1.

Table 2-1 WebLogic Server Security Standards Support

Standard	Version	Additional Considerations
JAAS	JAAS version depends on the Java SE version. See: • Java SE 17 - Java Authentication and Authorization Service (JAAS) in the *Java Security Developer's Guide* • Java SE 21 - Java Authentication and Authorization Service (JAAS) in the *Java Security Developer's Guide*	See Configuring a Domain to Use JAAS Authorization.
JASPIC	1.1	See Configuring JASPIC Security.
JACC	1.5	See Using the Java Authorization Contract for Containers.
Java EE application packaged permissions	Java EE 8 Platform Specification	
JCE	Jipher JCE 10.32 SunJCE	See Using JCE Providers with WebLogic Server.

Table 2-1 (Cont.) WebLogic Server Security Standards Support

Standard	Version	Additional Considerations
JSSE	Default SSL implementation based on Java Secure Socket Extension (JSSE).	See Using the JSSE-Based SSL Implementation **Note**: Although JSSE supports Server Name Indication (SNI) in its SSL implementation, WebLogic Server does not support SNI.
Kerberos	Version 5	See Configuring Single Sign-On with Microsoft Clients.
LDAP	v3	See: • Configuring LDAP Authentication Providers • Managing the Embedded LDAP Server
SAML	2.0	See: • Configuring SAML 2.0 Services
Security API (JSR 375)	1.0	See Using the Java EE Security API in WebLogic Server.
SLO	Via SAML	Supported by the Service Provider only. See Configure SAML Single Logout
SPNEGO	Specified by `https://datatracker.ietf.org/doc/html/rfc4178`.	See Configuring Single Sign-On with Microsoft Clients.
SSO	Via Microsoft Clients Via SAML	See: • Configuring Single Sign-On with Microsoft Clients • Configuring Single Sign-On with Web Browsers and HTTP Clients Using SAML.
TLS	v1.0, v1.1, v1.2, v1.3 **Note:** Support for TLS v1.0 and v1.1 is deprecated.	• TLS v1.2 is the default minimum protocol version configured in WebLogic Server. Oracle recommends the use of TLS v1.2 or later in a production environment. WebLogic Server logs a warning if the TLS version is set below 1.2. • Oracle strongly recommends that you do not use TLS v1.0 and v1.1. In addition, these versions may be disabled by default in certain JDK updates by the underlying JSSE provider. See Specifying the SSL/TLS Protocol Version for version-specific information.
Uncovered HTTP methods	Servlet 3.1	
X.509	v3	• WebLogic Server supports 4096-bit keys. (4096-bit keys may require substantially more compute time for some operations.) • Certificates generated with CertGen have a default 2048-bit key size. You specify the key size with the `-strength` option. • The WebLogic Server demo CA has a 2048-bit key length. • As of JDK 8, the use of X.509 certificates with RSA keys less than 1024 bits in length are blocked.

Table 2-1 (Cont.) WebLogic Server Security Standards Support

Standard	Version	Additional Considerations
xTensible Access Control Markup Language (XACML)	2.0	See Configuring Authorization and Role Mapping Providers.
Partial implementation of Core and Hierarchical Role Based Access Control (RBAC) Profile of XACML	2.0	Specified by http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf.

Supported FIPS Standards and Cipher Suites

WebLogic Server supports Federal Information Processing Standard (FIPS) publication 140-2 and cipher suites for JSSE JDK.

Table 2-2 lists the supported FIPS versions and cipher suites.

Table 2-2 Cipher Suites and FIPS 140-2 Supported Versions

Standard	Version	Additional Considerations
FIPS 140-2	Jipher	See Enabling FIPS Mode.
Cipher Suites for JSSE JDK 17	The preferred negotiated cipher combination is AES + SHA2.	To see the set of cipher suites supported by the JDK SunJSSE, see the *SunJSSE Provider* section in Java SE Security Developer's Guide.
Cipher suites supported in the (removed) WebLogic Server Certicom SSL implementation and the SunJSSE equivalent.	Product Dependent	Documented for backward compatibility. See Table 38-2. When using Certicom, WebLogic Server does not support SHA256 hashing, or signature algorithms that include SHA256.

3

Configuring Security for a WebLogic Domain

Configuring security for an Oracle WebLogic Server environment starts with a creating a secure installation of WebLogic Server. It also includes choosing the security configuration options that are appropriate for the environment in which the domain runs, such as obtaining and storing certificates, protecting user accounts, and securing the network on which the domain runs.

This chapter includes the following sections:

For a complete checklist of all components in the WebLogic Server that should be secured in a production environment, including specific tasks recommended for configuring a secure domain, securing the network, files and databases used by WebLogic Server, see Lock Down WebLogic Server in *Securing a Production Environment for Oracle WebLogic Server*.

Performing a Secure Installation of WebLogic Server

Performing a secure installation includes steps to secure the host machine on which WebLogic Server is installed, to limit access to that host to only authorized users, and to install Critical Patch Updates immediately after installation is complete.

If you are installing WebLogic Server in a production environment, Oracle strongly recommends the guidelines described in the following sections:

Before Installing WebLogic Server

Before you start the WebLogic Server installation program, complete the following tasks:

- Create a My Oracle Support account so that you can register your WebLogic Server installation with Oracle and receive security updates automatically. Visit `http://www.oracle.com/support/index.html`.

- Secure the host machine, operating system, and file system to ensure that access is restricted only to authorized users. For example:
  - Keep your hardware in a secured area to prevent unauthorized operating system users from gaining access to the machine and its network connections.
  - Make sure the host machine has the latest operating system patches and security updates.

    > **Note:**
    >
    > As new patches become available, you should download and install them promptly.

- Secure networking services and the file system that the operating system provides to prevent unauthorized access. For example, make sure that any file system sharing is secured.
- Set operating system file access permissions to restrict access to data stored on disk that will be used or managed by WebLogic Server, such as the security LDAP database and directories into which keystores are created and managed.
- Limit the number of user accounts on the host machine. Create a group to contain only the following user accounts:
  1. The user who installs WebLogic Server only.
  2. The user who creates the WebLogic domain and uses Node Manager to start the Administration Server and each Managed Server instance in the domain.

  Restrict the privileges of these user accounts to only the following directories:
  - Oracle home — Root directory created for all Oracle Fusion Middleware products on a host computer
  - WebLogic home — Root directory of the WebLogic Server installation
  - Domain home — Root directory of the WebLogic domain

    > **Note:**
    >
    > Some processes also need access to temporary directories by default, such as `/tmp` on Unix platforms. If the privileges of a user account are restricted to only the Oracle home, WebLogic home, and WebLogic domain directories, the user must change environment variables, such as `TEMP` or `TMP`, to point to a directory to which that user does have access.

- Ensure that any Web servers on the host machine run only as an unprivileged user, never as `root`.
- Ensure no software development tools or sample software is installed.
- Consider using additional software to secure your operating system, such as a reputable intrusion detection system (IDS).

See Secure the Host Environment in *Securing a Production Environment for Oracle WebLogic Server*.

While Running the Installation Program

During installation, make sure that you do not install the sample applications component.

Immediately After Installation is Complete

- Remove the Derby DBMS database, which is bundled with WebLogic Server for use by the sample applications and code examples as a demonstration database. Derby DBMS is located in the *WL_HOME*`/common/derby` directory.
- Visit the Critical Patch Updates, Security Alerts and Bulletins page at the following location to review WebLogic Server security advisories:

  https://www.oracle.com/security-alerts/

See the following topics in *Securing a Production Environment for Oracle WebLogic Server*:

- Read Security Publications
- Apply the Latest Patches and Updates

Creating a WebLogic Domain for Production Use

To create a WebLogic domain for production use, consider the environment in which the domain will run, such as whether it will interoperate with other WebLogic domains, and how best to secure the accounts of users who have access to the domain.

When you plan to use a WebLogic domain in a production environment, you should configure it in either production mode or secured production mode. The domain mode determines the default settings regarding security and logging.

In production mode, the security configuration is relatively stringent, such as requiring a user name and password to deploy applications and start the Administration Server. If you are using the `unpack` command to create a full WebLogic domain, or a subset of a domain that is used for a Managed Server domain directory on a remote machine, use the `-server_start_mode=prod` parameter to configure production mode.

In secured production mode, the default security configuration imposes more secure default values such as more restrictive authorization and role mapping policies, and logging warnings for insecure configuration settings in your domain. Note that in order to enable secured production mode, your domain must be in production mode.

> **Note:**
>
> By default, enabling production mode automatically enables *secured* production mode. If you want to enable production mode *without* the stricter settings of secured production mode, you must explicitly disable secured production mode.

Use one of the following tools to set domain mode:

- WebLogic Remote Console - see Change the Domain Mode in the *Oracle WebLogic Remote Console Online Help*
- WLST Offline - see the setOption WLST offline command in *WLST Command Reference for Oracle WebLogic Server*

- WLST Online - see Using WLST Online to Update an Existing WebLogic Domain in *Understanding the WebLogic Scripting Tool*
- Configuration Wizard - see Creating a WebLogic Domain in *Creating WebLogic Domains Using the Configuration Wizard*
- Pack/Unpack - see the `server_start_mode` parameter in The Pack Command in *Creating Templates and Domains Using the Pack and Unpack Commands*
- Fusion Middleware Control - see Configure domain security in *Administering Oracle WebLogic Server with Fusion Middleware Control*

If the domain will interoperate with other WebLogic domains, or has the potential for that use at some future point, choose resource names carefully. Many resource names are fixed at the time a domain is created, and stringent requirements must be observed for resource names when using cross-domain security, transactions, and messaging.

See Requirements for Transaction Communication in *Developing JTA Applications for Oracle WebLogic Server*.

When creating domains using WLST, do not enter unencrypted passwords in commands for configuring entities that require them, such as passwords for:

- Domain administrator
- Node Manager user
- Database user
- JKS and PKCS12 keystores (both when creating the keystores and again when configuring them with WebLogic Server)
- Wallet

Specifying unencrypted passwords in WLST commands is a security risk: they can be easily viewed from the monitor screen by others, and they are displayed in process listings that log the execution of those commands. Instead, omit the password from the command. When the command is executed, WLST automatically prompts you for any passwords needed to complete the domain configuration.

Securing the Domain After You Have Created It

After you have created your WebLogic domain, several key steps remain to ensure its integrity such as selecting an appropriate domain mode, limiting access to internal applications, and configuring a Password Validation provider. To secure a domain after you have created it, Oracle recommends the following steps:

1. Secure your production environment by selecting either production or secured production mode as the domain mode. When secured production mode is selected, attributes are set to their most secure value by default. However, regardless of domain mode, you can override an attribute's default value. It may be more appropriate for your environment to select production mode, and then specify secure values on applicable attributes only. To determine the domain mode that is most appropriate for your environment, see Understand How Domain Mode Affects the Default Security Configuration. WebLogic Server validates all security settings and logs warnings in case of insecure settings, thereby, providing a highly secure production environment. See Change the Domain Mode in *Oracle WebLogic Remote Console Online Help* to learn how to change your domain mode.
2. Limit access to internal applications by disabling unused internal applications using either the configuration settings or the system property. Enable the Administration port for your domain, and configure a firewall to prevent external access to internal applications on the Administration port. In secured production mode, the Administration port is enabled by

default. For information about how to disable internal applications and block access to them, see Disable Unused Internal Applications and Configure Firewall to Prevent Access to Internal Applications in *Securing a Production Environment for Oracle WebLogic Server*.

3. Configure the Password Validation provider to manage and enforce password composition rules. The Password Validation provider is configured out-of-the-box to work with several WebLogic authentication providers.

   See Configuring the Password Validation Provider.

4. As you create or add users to the security realm, check that the User Lockout options on user accounts are set for maximum protection. Note that the configuration of User Lockout is defined on a per realm basis. Therefore, if the default User Lockout settings are not suitable for your needs, you might need to customize these settings whenever you create a new security realm. See Protecting User Accounts and How Passwords Are Protected in WebLogic Server.

   If your domain is running in secured production mode, then WebLogic Server logs a warning if the user lockout is configured to a value less than the default value.

5. If you have configured Node Manager to start, shut down, and restart the Administration Server and Managed Server instances distributed across multiple machines, make sure that Node Manager security is properly configured.

   If you are using Java Node Manager (recommended for production environments), see Configuring Java-based Node Manager Security in *Administering Node Manager for Oracle WebLogic Server*.

   If you are using Script Node Manager, which may be suitable for environments that have less stringent security requirements, see Step 2: Configure Node Manager Security in *Administering Node Manager for Oracle WebLogic Server*.

6. Enable auditing, which provides an automated way of collecting and storing information about events and other activity occurring in the system. Auditing is available through either of the following means:

   - Configuration auditing — When this is enabled, the Administration Server emits log messages and generates audit events when a user changes the configuration of any resource within a domain or invokes management operations on any resource within a domain.
   - WebLogic Auditing provider — Optional security provider that collects, stores, and distributes information about operating requests and the outcome of those requests for the purposes of non-repudiation. When configuration auditing is enabled, the WebLogic Auditing provider also logs configuration auditing events.

   Note that auditing may impose a performance overhead that should be taken into consideration. However, by adjusting how auditing is configured, this additional overhead can be minimized. When enabling auditing, make sure that sufficient disk space is available for the audit log. See Configuring the WebLogic Auditing Provider.

> **Note:**
>
> If secured production mode is enabled for your domain, then WebLogic Server logs a warning if an Auditing provider is not configured. You can use the `WarnOnAuditing` attribute in the SecureModeMBean to specify whether warnings should be logged or not if auditing is not enabled.

7. Make sure that the JVM platform MBean server cannot be accessed remotely. See Monitoring and Management Using JMX Technology in *Java Monitoring and Management Guide*.
8. If you have a requirement to comply with Federal Information Processing Standards (FIPS) 140-2, complete the appropriate procedures described in Enabling FIPS Mode.
9. Make sure configuration settings for complete message time out are sized appropriately for your system. See Configuring Network Resources in *Administering Server Environments for Oracle WebLogic Server*.
10. Create and configure the keystores used for holding identity and trust; that is, the keystores containing identity certificates and the keystore containing trusted Certificate Authority (CA) certificates. See Configuring Keystores.

    If you are using the Oracle OPSS Keystore Service (KSS) for use with WebLogic Server, see Configuring Oracle OPSS Keystore Service.

    Configure certificate validation and revocation checking to ensure that:

    - Each certificate in a certificate chain was issued by a certificate authority, as explained in SSL Certificate Validation.
    - The revocation status of each certificate WebLogic Server validates is current. See X.509 Certificate Revocation Checking.
11. Configure a host name verifier. When making an SSL connection, the host name verifier ensures that the host name in the URL to which the client connects matches the host name in the digital certificate that the server sends back. See Using Host Name Verification.

    If your domain is running in secured production mode, then WebLogic Server logs a warning if host name verification is disabled. To enable host name verification, see Enable Host Name Verification in *Oracle WebLogic Remote Console Online Help*.
12. Configure TLS/SSL for the administration port, network channels, database connections, LDAP server connections, and other resources handling communication that must be secured. In particular, make sure that connections to remote server instances in the domain are secured with TLS/SSL. The specific components for which either one- or two-way TLS/SSL needs to be configured depends on the overall topology of the production environment. See the following topics:

Table 3-1 TLS/SSL Configuration Topics

For information about . . .	See the following topic . . .
An overview of using SSL to secure communications in a basic WebLogic domain	Secure Sockets Layer (SSL) in *Understanding Security for Oracle WebLogic Server*
Where to use one-way and two-way SSL in a basic WebLogic domain	One-way/Two-way SSL Authentication in *Understanding Security for Oracle WebLogic Server*
Steps to configure SSL in a basic WebLogic domain	Setting Up SSL/TLS: Main Steps
Configuring an administration port for secure communication with the domain Administration Server	Administration Port and Administrative Channel in *Administering Server Environments for Oracle WebLogic Server*
Securing database connections	Understanding Data Source Security in *Administering JDBC Data Sources for Oracle WebLogic Server*

Table 3-1 (Cont.) TLS/SSL Configuration Topics

For information about . . .	See the following topic . . .
Best practices for configuring SSL in WebLogic Server	"Section 2. Security Best Practices" in Document ID 1074055.1, available from My Oracle Support at https://support.oracle.com/

> **Note:**
>
> Note the following:
>
> - By default, WebLogic Server is configured for one-way SSL authentication; however, the SSL port is disabled. Oracle strongly recommends enabling the SSL port in all server instances in a production domain.
> - The demonstration digital certificates, private keys, and trusted CA certificates provided in WebLogic Server should never be used in a production environment.
> - In secured production mode, WebLogic Server logs warnings if the SSL configuration is not secure. You can use the `WarnOnInsecureSSL` attribute contained in the SecureModeMBean to specify whether warnings should be logged if the SSL configuration is not secure.

13. Restrict the size and the time limit of requests on external channels to prevent Denial of Service attacks. See Reducing the Potential for Denial of Service Attacks in *Tuning Performance of Oracle WebLogic Server*.
14. If you use multiple Authentication providers, be sure to set the JAAS control flag correctly. See Using More Than One Authentication Provider.
15. Ensure that you have correctly assigned users and groups to the default WebLogic Server security roles. See Users, Groups, And Security Roles in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*.
16. Review the Security Warnings Report for any outstanding security validation issues in your domain. See Review Potential Security Issues in *Securing a Production Environment for Oracle WebLogic Server*.

Obtaining Private Keys, Digital Certificates, and Trusted Certificate Authority Certificates

You have multiple choices for obtaining private keys, digital certificates, and trusted CA certificates for your WebLogic Server environment. Oracle strongly recommends obtaining private keys and digital certificates from a reputed certificate authority. When choosing these items, note the following considerations:

- For production environments, Oracle strongly recommends obtaining private keys and digital certificates only from a reputable certificate authority such as Entrust or Symantec Corporation. See Obtaining and Storing Certificates for Production Environments.
- For development environments only, you can use the digital certificates, private keys, and trusted CA certificates provided by WebLogic Server. You can also use keytool or the CertGen utility to generate self-signed certificates. See Using Keystores and Certificates in a Development Environment.

Storing Private Keys, Digital Certificates, and Trusted Certificate Authority Certificates

Once you have obtained private keys, digital certificates, and trusted CA certificates, you need to store them so that WebLogic Server can use them to find and verify identity. Private keys, their associated digital certificates, and trusted CA certificates are stored in keystores. Then you need to configure those keystores with WebLogic Server.

For information about . . .	See the following topic . . .
Creating a keystore	Creating a Keystore
Configuring a keystore to be used with WebLogic Server	Configuring Keystores with WebLogic Server
A step-by-step example of using the keytool utility to create a keystore and store keys and certificates in it	Creating a Keystore: An Example
Displaying the certificates contained in a keystore	Viewing Keystore Contents
Updating certificates that are due to expire	Replacing Expiring Certificates
Setting reminders about certificate expiration	Setting Certificate Expiry Notifications

Protecting User Accounts

WebLogic Server provides a set of configuration options to protect user accounts from intruders. In the default security configuration, these options are set for maximum protection. You can use WebLogic Remote Console to modify these options for each security realm. As a system administrator, you have the option of turning off all the configuration options, increasing the number of login attempts before a user account is locked, increasing the time period in which invalid login attempts are made before locking the user account, and changing the amount of time a user account is locked.

For information about the User Lockout Manager MBean, see UserLockoutManagerMBean in *MBean Reference for Oracle WebLogic Server*.

Remember that changing the configuration options lessens security and leaves user accounts vulnerable to security attacks. See Set User Lockout Attributes in *Oracle WebLogic Remote Console Online Help*.

> **Note:**
>
> The User Lockout options apply to the default security realm and all its security providers. User Lockout works in all security realms, is layered on top of all configured providers, including custom ones, and is enabled by default.
>
> If you are using an Authentication provider that has its own mechanism for protecting user accounts, consider if disabling User Lockout on the security realm is appropriate because other Authentication providers might be configured in the security realm.
>
> If a user account becomes locked and you delete the user account and add another user account with the same name and password, the User Lockout configuration options will not be reset.

For information about unlocking a locked user account, see Unlock a User in *Oracle WebLogic Remote Console Online Help*. Unlocking a locked user account can be done through either WebLogic Remote Console or the `clearLockout` attribute on the `UserLockoutManagerRuntimeMBean`. See UserLockoutManagerRuntimeMBean in *MBean Reference for Oracle WebLogic Server*.

Using Connection Filters

Connection filters allow you to deny access at the network level. They can be used to protect server resources on individual servers, server clusters, or an entire internal network or intranet. For example, you can deny any non-SSL connections originating outside of your corporate network. Network connection filters are a type of firewall in that they can be configured to filter on protocols, IP addresses, and DNS node names.
Connection filters are particularly useful when using the Administration port. Depending on your network firewall configuration, you may be able to use a connection filter to further restrict administration access. A typical use might be to restrict access to the Administration port to only the servers and machines in the WebLogic domain. An attacker who gets access to a machine inside the firewall, still cannot perform administration operations unless the attacker is on one of the permitted machines.

WebLogic Server provides a default connection filter called `weblogic.security.net.ConnectionFilterImpl`. This connection filter accepts all incoming connections and also provides static factory methods that allow the server to obtain the current connection filter. To configure this connection filter to deny access, simply enter the connection filters rules in WebLogic Remote Console.

You can also use a custom connection filter by implementing the classes in the `weblogic.security.net` package. For information about writing a connection filter, see Using Network Connection Filters in *Developing Applications with the WebLogic Security Service*. Like the default connection filter, custom connection filters are configured in WebLogic Remote Console.

To configure a connection filter:

1. Enable the logging of accepted messages. This Connection Logger Enabled option logs successful connections and connection data in the server. This information can be used to debug problems relating to server connections.
2. Choose which connection filter is to be used in the domain.
   - To configure the default connection filter, specify `weblogic.security.net.ConnectionFilterImpl` in Connection Filter.
   - To configure a custom connection filter, specify the class that implements the network connection filter in Connection Filter. This class must also be specified in the `CLASSPATH` for WebLogic Server.
3. Enter the syntax for the connection filter rules.

For more information, refer to the following topics:

- Configure Connection Filtering in *Oracle WebLogic Remote Console Online Help*
- Using Network Connection Filters and Developing Custom Connection Filters in *Developing Applications with the WebLogic Security Service*.

You can also use the WebLogic Scripting Tool (WLST) or Java Management Extensions (JMX) APIs to create a new security configuration.

Using JEP 290 in Oracle WebLogic Server

To improve security, WebLogic Server uses the JDK JEP 290 mechanism to filter incoming serialized Java objects and limit the classes that can be deserialized. The filter helps to protect against attacks from specially crafted, malicious serialized objects that can cause denial of service (DOS) or remote code execution (RCE) attacks.

There are two models to prevent deserialization exploits: blocklist and allowlist. With the blocklist model, WebLogic Server defines a set of well-known classes and packages that are vulnerable and blocks them from being deserialized and all other classes can be deserialized. In the allowlist model, WebLogic Server and the customer define a list of the acceptable classes and packages that are allowed to be deserialized, and block all other classes. While both approaches have benefits, the allowlist model is more secure because it only allows deserialization of classes and packages known to be required by WebLogic Server and customer applications.

You have the option of choosing whether to use blocklists or allowlists.

- WebLogic Server uses blocklists by default. At startup, WebLogic Server configures a default JEP 290 blocklist filter that specifies the maximum depth of a graph and a set of prohibited classes and packages that cannot be deserialized. You can then use WebLogic Server JEP 290 properties to customize the blocklist to add additional classes or packages. You can also use dynamic blocklists, which provide the ability to update your blocklist filters by creating configuration files that can be updated or replaced while the server is running. See Using Dynamic Blocklist Configuration Files.

  > **Note:**
  >
  > These default blocklist settings, including the set of prohibited classes specified in the default filter, can change over time. WebLogic Server Patch Set Updates (PSUs) may include updates to the set of prohibited classes and packages used in the default filter. To ensure that your system is protected with the most current default filter, be sure to apply the latest WebLogic Server PSUs and Java Critical Patch Updates (CPUs) as soon as they are released. The Critical Patch Updates, Security Alerts and Bulletins page references the latest Java and WebLogic Server updates that are available on My Oracle Support.

- Alternatively, you can choose to use allowlists. First, you must create an allowlist that contains the classes and packages that are deserialized in the applications in your domain. To do so, you enable recording, which records all of the classes and packages used in both WebLogic Server and customer application deserialization. When deserialization occurs, each class is recorded in an allowlist configuration file. When you are satisfied with the allowlist, you then configure WebLogic Server to use the allowlist configuration file for the JEP 290 filtering. See Using an Allowlist for JEP 290 Filtering.

JEP 290 filter syntax supports both the blocklist and allowlist models. For JEP 290 filter syntax, see the Process-wide Filter section in `http://openjdk.java.net/jeps/290`. The configuration files that WebLogic Server uses to either block or allow classes adhere to the JEP 290 syntax. For example, a pattern `!foo.bar.Mumble` blocks the class `foo.bar.Mumble`. Classes and packages that are not preceded by the `!` are allowed.

How WebLogic Server Uses JEP 290 Blocklists and Allowlists

WebLogic Server uses both JEP 290 blocklist and allowlist models to prevent deserialization exploits.

If you are using the blocklist model, WebLogic Server uses JEP 290 as follows:

- Implements a WebLogic Server-specific JEP 290 filter to enforce a blocklist of prohibited classes and packages for deserialization used by WebLogic Server, and allows all other classes and packages. The filter also enforces a default value for the maximum depth of a deserialized object tree.
- Implements a global JEP 290 filter to enforce a blocklist of prohibited classes and packages for deserialization used by applications or third party libraries.
- Provides WebLogic Server JEP 290 properties that you can use to customize the default filter such as adding or removing classes and packages from the default filter to block particular classes.

If you are using the allowlist model, WebLogic Server uses JEP 290 as follows:

- Implements a WebLogic Server-specific JEP 290 filter to enforce an allowlist of allowed classes and packages for deserialization used by WebLogic Server, and blocks all other classes and packages.
- Implements a global JEP 290 filter to enforce an allowlist of allowed classes and packages for deserialization used by applications or third party libraries, and blocks all other classes and packages.
- Provides WebLogic Server JEP 290 properties that you can use to customize the allowlist, such as adding or removing classes and packages from the recorded filter to allow or block particular classes.

You can also use the JEP 290 properties to filter deserialized classes based on the nesting depth of the deserialized object, the number of internal references in the deserialized object, the size of arrays, and/or the maximum size in bytes of a deserialized object.

Customizing JEP 290 Filters Using Properties

WebLogic Server includes properties that you can use to customize, replace, or disable the JEP 290 filters if desired. These properties can be specified on the command line as system properties or contained as properties in the JEP 290 dynamic configuration and allowlist configuration files.

For the latest information on the WebLogic Server JEP 290 default blocklist filter, see the My Oracle Support document *Restricting Incoming Serialized Java Objects to Oracle WebLogic Server (Doc ID 2421487.1)*.

The following table describes the properties and includes sample usage.

Table 3-2 WebLogic Server JEP 290 Properties

Property	Description
`weblogic.oif.serialFilter`	Use this property to set a custom JEP 290 filter for WebLogic Server, using the standard JEP 290 filter syntax. For JEP 290 filter syntax, see the Process-wide Filter section in http://openjdk.java.net/jeps/290. By default, this custom filter is combined with the default WebLogic Server filter, with the custom filter taking precedence over the default filter for any filter elements that conflict. If you are using the allowlist model, all blocked classes and packages are given the highest priority in the allowlist filter. For example, to set a custom filter by adding a class named `foo.bar.Mumble` to the default blocklist, use: `-Dweblogic.oif.serialFilter="!foo.bar.Mumble"` This setting blocks the class `foo.bar.Mumble` even if it is allowed by the default filter.
`weblogic.oif.serialFilterMode`	Use this property to specify the filter mode for the custom filter, which provides the ability to combine, replace, or disable the default WebLogic Server filter. Valid values are: • `combine` — combines the custom filter with the default WebLogic Server filter. The custom filter settings take precedence over the default filter settings for any filter elements that conflict. This is the default. If you are using the allowlist model, all blocked classes and packages are given the highest priority in the allowlist filter. • `replace` — replaces the default WebLogic Server filter with the custom filter. Oracle recommends that you include all of the blocklist and allowlist classes and packages from the default WebLogic Server filter in your replacement filter. If you do not include them, then your system will not be protected from malicious deserialization attacks. • `disable` — disables the default WebLogic Server filter. Oracle strongly recommends that you do not disable the filter. If you do so, then your system will not be protected from malicious deserialization attacks. For example, to replace the default WebLogic Server filter with the custom filter, use: `-Dweblogic.oif.serialFilterMode=replace`
`weblogic.oif.serialFilterScope`	Use this property to specify whether the filter should apply globally to the entire JVM (customer application and third-party library deserialization) or to only internal WebLogic Server deserialization. Valid values are `global` and `weblogic`. The default is `global`. For example, to apply the WebLogic Server default or custom filter to internal WebLogic Server deserialization only, instead of to the entire JVM, use: `-Dweblogic.oif.serialFilterScope=weblogic`

Table 3-2 (Cont.) WebLogic Server JEP 290 Properties

Property	Description
`weblogic.oif.serialGlobalFilter`	Use this property to set a custom JEP 290 global filter for WebLogic Server, using the standard JEP 290 filter syntax. For JEP 290 filter syntax, see the Process-wide Filter section in `http://openjdk.java.net/jeps/290`. By default, this custom global filter is combined with the default WebLogic Server filter, with the custom global filter taking precedence over the default filter for any filter elements that conflict. This global filter applies to object input streams used for application and third party library deserialization, and does not apply to WebLogic Server deserialization For example, to set a custom global filter by adding a class named `foo.bar.Mumble` to the default blocklist, use: `-Dweblogic.oif.serialGlobalFilter="!foo.bar.Mumble"` This setting blocks the class `foo.bar.Mumble` from deserialization in customer applications and third party libraries, even if it is allowed by the default filter.
`weblogic.oif.head.serialFilter`	Use this property to override a custom JEP 290 filter for WebLogic Server, using the standard JEP 290 filter syntax. For JEP 290 filter syntax, see the Process-wide Filter section in `http://openjdk.java.net/jeps/290`. By default, this head filter is combined with the custom and default WebLogic Server filters, with the head filter taking precedence over both the custom and default filter for any filter elements that conflict. For example, to set a head filter by adding a class named `foo.bar.Mumble` to the allowlist, use: `-Dweblogic.oif.head.serialFilter="!foo.bar.Mumble"` This setting blocks the class `foo.bar.Mumble` from WebLogic Server deserialization even if it is allowed by the custom and default filters.
`weblogic.oif.head.serialGlobalFilter`	Use this property to override a custom JEP 290 global filter for WebLogic Server, using the standard JEP 290 filter syntax. For JEP 290 filter syntax, see the Process-wide Filter section in `http://openjdk.java.net/jeps/290`. By default, this custom global filter is combined with the custom and default WebLogic Server filters, with the head global filter taking precedence over both the custom and default filter for any filter elements that conflict. This global filter applies to object input streams used for application and third party library deserialization, and does not apply to WebLogic Server deserialization For example, to set a head global filter by adding a class named `foo.bar.Mumble` to the allowlist, use: `-Dweblogic.oif.head.serialGlobalFilter="!foo.bar.Mumble"` This setting blocks the class `foo.bar.Mumble` from deserialization in customer applications and third party libraries even if it is allowed by the custom and default filters.

Table 3-2 (Cont.) WebLogic Server JEP 290 Properties

Property	Description
`weblogic.oif.serialUnauthenticatedFilter`	Use this property to set a custom JEP 290 filter for the unauthenticated code path of WebLogic Server, using the standard JEP 290 filter syntax. For JEP 290 filter syntax, see the Process-wide Filter section in `http://openjdk.java.net/jeps/290`. **Note:** This filter is used when you have disabled remote anonymous RMI T3 and IIOP requests. Oracle does not expect that users will need to customize the unauthenticated code path filter. The current set of allowed classes during the unauthenticated code path should be sufficient. By default, this custom filter is combined with the default WebLogic Server unauthenticated filter, with the custom filter taking precedence over the default filter for any filter elements that conflict. For example, to set a custom filter by adding a class named `foo.bar.Mumble` to the default unauthenticated allowlist, use: `-Dweblogic.oif.serialUnauthenticatedFilter="foo.bar.Mumble"` This setting allows the class `foo.bar.Mumble` to be used in the unauthenticated code path.

Using Dynamic Blocklist Configuration Files

Dynamic blocklists provide the ability to update your blocklist filters by creating configuration files that can be updated or replaced while the server is running.

By default, WebLogic Server will detect the presence of a dynamic blocklist configuration file located in the *DOMAIN_HOME*`/config/security` directory, and block deserialization of classes specified in the configuration file.

WebLogic Server can locate dynamic blocklist configuration files that you place in other directories, for example the Oracle Home directory, and block deserialization using those files as appropriate. For WebLogic Server to detect the presence of these files, you must specify the locations of the files using a new system property, `weblogic.oif.serialPropDirectories`, and include the property in the WebLogic Server start-up script.

WebLogic Server can also detect the presence of a dynamic blocklist configuration file in the *ORACLE_HOME*`/oracle_common/common/jep290` directory, and block deserialization of classes and packages specified in the file.

When blocklist files are saved in these locations, WebLogic Server reads them at the specified time interval and immediately begins enforcing the blocks that are specified. You can update or replace the files without needing to stop the server.

To use dynamic blocklists:

1. Create a configuration file that contains the desired WebLogic, global and unauthenticated filters and save it using the suffix `serial.properties`. Ensure that the filter strings do not contain any white spaces. Also ensure that WebLogic Server has read permission to the configuration file or server start up will fail. A sample `serial.properties` file is shown here:

```
weblogic.oif.serialFilter=\
!MyCustomer1.Employee;\
!MyCustomer2.Employee2;\
!MyCustomer3.*;\
!MyCustomer4.**;\
!MyCustomer5.Employee5

weblogic.oif.serialGlobalFilter=\
!MyCustomer1.Employee;\
!MyCustomer2.Employee2;\
!MyCustomer3.**;\
!MyCustomer4.**;\
!MyCustomer5.Employee5
```

   In this example:

   - The `weblogic.oif.serialFilter` applies to WebLogic Server deserialization.
   - The `weblogic.oif.serialGlobalFilter` applies to customer application and third party library deserialization.

   For both of these filters, the first one matched takes precedence over the others in the filter. For a description of these filters, see Table 3-2.

2. To use the default location, save the file to the *`DOMAIN_HOME`*`/config/security` directory. In this pathname, *`DOMAIN_HOME`* represents the WebLogic domain root directory. WebLogic Server locates the file in this directory by default.

   If you are not using the default location, save the file with the suffix `serial.properties` to the desired directory and specify the directory location using the `weblogic.oif.serialPropDirectories` system property in the startup script. You can specify multiple files and locations. For example:

   ```
   -Dweblogic.oif.serialPropDirectories=/u01/oracle/fmw/app1:/u01/oracle/fmw/app2
   ```

3. By default, these directories are polled every 60 seconds. To change the default polling interval, set the `weblogic.oif.serialPropPollingFileInterval` system property in the startup script. For example, to set the polling interval to 10 seconds, use:

   ```
   -Dweblogic.oif.serialPropPollingFileInterval=10000
   ```

Using an Allowlist for JEP 290 Filtering

Allowlists are configuration files that define a list of the WebLogic Server and customer application classes and packages that you wish to allow to be deserialized. Allowlists can be created and configured to control which packages and classes are deserialized (or blocked) in running systems.

To create and configure a customer allowlist:

1. In a staging or test environment, enable recording using either of the following methods:
   - Use WebLogic Remote Console:

a. In the **Edit Tree**, go to **Environment**, then **Domain**.
b. On the **Security** tab, select the **Allow List** subtab.
c. Turn on the **Recording Enabled** option.
d. Click **Save** and then commit your changes.

- Use WLST online to set the `AllowListRecordingEnabled` attribute on the `AllowListMBean`:

```
edit()
startEdit()
cd("AllowList/mydomain")
cmo.setAllowListRecordingEnabled(true)
save()
activate()
disconnect()
```

When recording is enabled, all classes are allowed during deserialization except for the classes specified in the blocklist.

2. Run a full set of tests to ensure that the recorded allowlist configuration file provides appropriate coverage of all packages and classes that must be allowed in order for your application to run successfully. When deserialization occurs, each class is recorded in the following configuration file:

   DOMAIN_HOME`/config/security/jep290-recorded.serial.properties`

   In this pathname, *DOMAIN_HOME* represents the WebLogic domain root directory.

   A sample `jep290-recorded.serial.properties` is shown here:

```
Wed May 19 23:55:13 UTC 2021
weblogic.oif.serialFilter=\
    com.company1.common.collections.objs.*;\
    com.company1.common.tools.Calculator;\
    com.company2.shared.tools.Converter
weblogic.oif.serialGlobalFilter=\
    com.company1.common.lists.AList;\
    com.company1.common.tools.Calculator;\
    com.company2.shared.tools.*
```

3. Turn off recording using either WebLogic Remote Console or WLST online:
   - In WebLogic Remote Console, turn off the **Recording Enabled**option. Click **Save**, then commit your changes.
   - Use WLST online to set the `AllowListRecordingEnabled` attribute to `false`.
4. Configure the WebLogic Server domain to use either allowlists or blocklists in one of the following ways:
   - In WebLogic Remote Console, in the **Edit Tree**, go to **Environment**, then **Domain**. Click the **Security** tab, then the **Allow List** subtab. From the **Violation Action** drop-down list, select the desired setting. Click **Save** and commit your changes.
   - Use WLST online to set the `AllowListViolationAction` attribute on the `AllowListMBean`.

   The available settings are as follows:

- `IGNORE` - Ignore the allowlist and use the blocklists. If any class found during deserialization is present in the blocklist, the class is blocked from being deserialized.
- `DENY` - Block everything except the classes specified in the allowlist, and log a message when a class is blocked.
- `LOG` - Log a message if a violation occurs but allow the class unless it is listed in the blocklist.

> **Note:**
>
> You can also set the `AllowListViolationAction` on a channel using the network access point. Doing so allows you to use an allowlist on untrusted external channels and a blocklist on internal trusted channels.

5. By default, the directory containing the allowlist configuration file is polled every 60 seconds. To change the default polling interval, do one of the following:
   - In WebLogic Remote Console, in the **Edit Tree**, go to **Environment**, then **Domain**. Click the **Security** tab, then the **Allow List** subtab. In the **Serial Profile Polling Interval** field, enter the new desired interval. Click **Save** and then commit your changes.
   - Set the `serialPropPollingFileInterval` attribute on the `AllowListMBean` to the desired interval.
   - Set the `weblogic.oif.serialPropPollingFileInterval` system property in the startup script. For example, to set the polling interval to 10 seconds, use:

     ```
     -Dweblogic.oif.serialPropPollingFileInterval=10000
     ```
6. Configure your production domain to use allowlists by copying the recorded allowlist configuration file that you created in Step 2 to the *`DOMAIN_HOME`*`/config/security` directory of the production domain.

> **Note:**
>
> Oracle recommends that you run your production domain with `AllowListViolationAction` set to `Log` for some period of time to ensure that all classes and packages were recorded.

7. Maintain the accuracy of the allowlist configuration file. Whenever a new application is deployed to the domain, or a new version of the application is deployed, you should repeat this process, beginning at Step 1, to recreate the allowlist or verify the allowlist with the new application to ensure that all packages and classes required by the new or updated application are included in the allowlist.

Customizing the Allowlist After Recording

Oracle recommends that you use the recording method to create an allowlist whenever possible. If customization is required after creating the allowlist configuration file, then:

1. Turn off recording.
2. Make a backup copy of the `jep290-recorded.serial.properties` file.

3. Edit the `jep290-recorded.serial.properties` in the *`DOMAIN_HOME`*`/config/security/` directory to add or remove classes as required. See Customizing JEP 290 Filters Using Properties.

This list provides examples for editing the following sample `jep290-recorded.serial.properties` file:

```
Wed May 19 23:55:13 UTC 2021
weblogic.oif.serialFilter=\
    com.company1.common.collections.objs.*;\
    com.company1.common.tools.Calculator;\
    com.company2.shared.tools.Converter
weblogic.oif.serialGlobalFilter=\
    com.company1.common.lists.AList;\
    com.company1.common.tools.Calculator;\
    com.company2.shared.tools.*
```

- **Removing a class from the recorded file**

  If the recorded allowlist file is allowing a class that you want to block, then edit the recorded `jep290-recorded.serial.properties` file to remove the class. For example, to remove the class `com.company1.common.tools.Calculator;` from both the WebLogic and global filters in the sample recorded allowlist file, remove the row from both the `weblogic.oif.serialFilter=\` and `weblogic.oif.serialGlobalFilter=\` stanzas.

  The resulting sample file is as follows:

  ```
  Wed May 19 23:55:13 UTC 2021
  weblogic.oif.serialFilter=\
      com.company1.common.collections.objs.*;\
      com.company2.shared.tools.Converter
  weblogic.oif.serialGlobalFilter=\
      com.company1.common.lists.AList;\
      com.company2.shared.tools.*
  ```

- **Adding a class to the recorded allowlist file**

  If you want to add a class to the recorded allowlist file this is being blocked, then edit the recorded `jep290-recorded.serial.properties` file to add the class to the desired filter. For example, to add the class `com.company1.MyApplication1` to both the WebLogic and global filters in the sample recorded file, add the row to both the `weblogic.oif.serialFilter=\` and `weblogic.oif.serialGlobalFilter=\` stanzas.

  The resulting sample file is as follows (the bold text identifies the added content):

  ```
  Wed May 19 23:55:13 UTC 2021
  weblogic.oif.serialFilter=\
      com.company1.common.collections.objs.*;\
      com.company1.common.tools.Calculator;\
      com.company2.shared.tools.Converter;\
      com.company1.MyApplication1
  weblogic.oif.serialGlobalFilter=\
      com.company1.common.lists.AList;\
      com.company1.common.tools.Calculator;\
      com.company2.shared.tools.*;\
      com.company1.MyApplication1
  ```

- **Allowing a class that WebLogic Server is blocking**

  As described in Understanding the Filter Order Preference, all blocked classes and packages are given the highest priority in the allowlist filter unless they are specifically allowed by the `weblogic.oif.head.serialGlobalFilter` or `weblogic.oif.head.serialFilter` properties.

  Therefore, if the class `org.codehaus.groovy.runtime.ConvertedClosure` is blocked by the WebLogic Server custom or default filter and you want to allow this class for global object input streams (used for application and 3rd party library deserialization), you can use the `weblogic.oif.head.serialGlobalFilter` JEP 290 property to override the setting in the filter. You can do so using either of the following methods:

  – Specify the following property on the command line as a system property:

    ```
    -
    Dweblogic.oif.head.serialGlobalFilter=org.codehaus.groovy.runtime.ConvertedClosur
    e
    ```

  – Add the following lines to the sample recorded *DOMAIN_HOME*`/config/security/jep290-recorded.serial.properties` file:

    ```
    weblogic.oif.head.serialGlobalFilter=\
        org.codehaus.groovy.runtime.ConvertedClosure
    ```

    The resulting sample allowlist file is as follows (the bold text identifies the added content):

    ```
    Wed May 19 23:55:13 UTC 2021
    weblogic.oif.serialFilter=\
        com.company1.common.collections.objs.*;\
        com.company1.common.tools.Calculator;\
        com.company2.shared.tools.Converter
    weblogic.oif.serialGlobalFilter=\
        com.company1.common.lists.AList;\
        com.company1.common.tools.Calculator;\
        com.company2.shared.tools.*
    weblogic.oif.head.serialGlobalFilter=\
        org.codehaus.groovy.runtime.ConvertedClosure
    weblogic.oif.serialUnauthenticatedFilter=\
    ```

  If the class `org.codehaus.groovy.runtime.ConvertedClosure` is blocked by the WebLogic Server custom or default filter and you want to allow this class for WebLogic Server deserialization, you can do either of the following:

  – Specify the following property on the command line as a system property:

    ```
    -Dweblogic.oif.head.serialFilter=org.codehaus.groovy.runtime.ConvertedClosure
    ```

  – Add the following lines to the sample recorded *DOMAIN_HOME*`/config/security/jep290-recorded.serial.properties` file:

    ```
    weblogic.oif.head.serialFilter=\
        org.codehaus.groovy.runtime.ConvertedClosure
    ```

The resulting sample file is as follows (the bold text identifies the added content):

```
Wed May 19 23:55:13 UTC 2021
weblogic.oif.serialFilter=\
    com.company1.common.collections.objs.*;\
    com.company1.common.tools.Calculator;\
    com.company2.shared.tools.Converter
weblogic.oif.serialGlobalFilter=\
    com.company1.common.lists.AList;\
    com.company1.common.tools.Calculator;\
    com.company2.shared.tools.*
weblogic.oif.head.serialFilter=\
    org.codehaus.groovy.runtime.ConvertedClosure
weblogic.oif.serialUnauthenticatedFilter=\
```

After you edit the file as required and save it, the server picks up the edited file after the specified polling interval.

Enabling Filter Logging

WebLogic Server provides a system property, `weblogic.oif.serialFilterLogging`, and a debug flag, `DebugAllowList` on the `ServerDebugMBean`, that you can use to log the current blocklist and allowlist classes and packages.

To enable logging, start WebLogic Server with the `weblogic.oif.serialFilterLogging` system property set to `true`. For example:

```
./startWebLogic.sh -Dweblogic.oif.serialFilterLogging=true
```

To get more implementation specific logging details, you can use the `DebugAllowList` attribute in `ServerDebugMBean`. Set this property to `true` in the WebLogic Remote Console , using WLST, or on the command line, for example `-Dweblogic.debug.DebugAllowList=true`.

The following log shows sample output using the system property. The filter settings, as well as the blocklist and allowlist classes and packages, are displayed in the server log.

```
<Jul 1, 2021 9:07:33,787 PM UTC> <Info> <WebLogicServer> <BEA-003807> <The
WebLogic Server JEP 290 filter mode is COMBINE>
<Jul 1, 2021 9:07:33,787 PM UTC> <Info> <WebLogicServer> <BEA-003808> <The
WebLogic Server JEP 290 filter scope is GLOBAL>
<Jul 1, 2021 9:07:33,788 PM UTC> <Info> <WebLogicServer> <BEA-003810>
<WebLogic Server JEP 290 filter limit element for scope WEBLOGIC is:
maxdepth=250>
...
<Jul 1, 2021 9:07:33,790 PM UTC> <Info> <WebLogicServer> <BEA-003811>
<WebLogic Server JEP 290 filter blocklist package for scope WEBLOGIC is:
org.apache.commons.collections.functors>
...
<Jul 1, 2021 9:07:33,802 PM UTC> <Info> <WebLogicServer> <BEA-003812>
<WebLogic Server JEP 290 filter blocklist class for scope WEBLOGIC is:
java.rmi.server.RemoteObject>
...
<Jul 1, 2021 9:07:33,806 PM UTC> <Info> <WebLogicServer> <BEA-003813>
<WebLogic Server JEP 290 filter allowlist for scope WEBLOGIC is: weblogic.**>
<Jul 1, 2021 9:07:33,806 PM UTC> <Info> <WebLogicServer> <BEA-003813>
```

```
<WebLogic Server JEP 290 filter allowlist for scope WEBLOGIC is: oracle.**>
...
<Jul 1, 2021 9:07:33,827 PM UTC> <Info> <WebLogicServer> <BEA-003813>
<WebLogic Server JEP 290 filter allowlist for scope GLOBAL is: java.**>
...
```

> **Note:**
>
> The filter log also displays the filters being used, and any additions or deletions. The filter string for each type of filter used, such as `weblogic.oif.serialFilter`, `weblogic.oif.serialGlobalFilter`, and `weblogic.oif.serialUnauthenticatedFilter`, shows the order of blocklist and allowlist entries in the filter.

Understanding the Filter Order Preference

WebLogic Server combines the blocked and allowed classes and packages specified using properties and configuration files to create a filter that is used to determine order preference for the blocklists and allowlists.

The order preference for the filter created, from highest priority to lowest priority, is determined as follows:

- Custom filters specified at server startup using the `weblogic.oif.head.serialFilter` and `weblogic.oif.head.serialGlobalFilter` properties (highest).
- Custom filters specified at server startup using the `weblogic.oif.serialFilter` and `weblogic.oif.serialGlobalFilter` properties.
- Custom filters specified in configuration files using the `weblogic.oif.serialPropDirectories` property.
- Custom filters specified in a configuration file located in the default directory *`DOMAIN_HOME`*`/config/security`.
- Custom filters specified in a configuration file located in the Oracle home directory *`ORACLE_HOME`*`/oracle_common/common/jep290`.
- The WebLogic Server default filter (lowest).

> **Note:**
>
> If you are using allowlists, all blocked classes and packages are given the highest priority in the allowlist filter unless they are specifically allowed by the `weblogic.oif.head.serialFilter` or `weblogic.oif.head.serialGlobalFilter` properties.

Setting the Deserialization Timeout Interval

You can further strengthen your protection against potential denial of service attacks by setting a time limit on deserialization. When the time limit elapses, the deserialization process is automatically terminated.

WebLogic Server adds support for the KernelMBean attribute `RMIDeserializationMaxTimeLimit` and the `weblogic.rmi.stream.deserialization.timelimitmillis` system property to configure the deserialization time limit.

By default, the time limit is disabled and not enforced when deserializing Java objects. To add a limit, set your desired time interval, in milliseconds, using either the `RMIDeserializationMaxTimeLimit` attribute in the KernelMBean or the `weblogic.rmi.stream.deserialization.timelimitmillis` system property. WebLogic Server does not apply a single interval across multiple Java objects. Instead each top-level object triggers its own separate time limit.

For example, to set a time limit of 10 seconds, use `-Dweblogic.rmi.stream.deserialization.timelimitseconds=10000`.

Enter an interval of 100 ms or longer. Very short intervals may prevent deserialization from operating smoothly.

Enter 0 to disable the time limit.

> **Note:**
>
> The deserialization time limit will not take effect if the JEP 290 system property `weblogic.oif.serialFilterMode` is set to `disable`.

JTA TransactionLoggable Allowlist

The JTA TransactionLoggable allowlist is added to address a potential vulnerability with the JTA transaction log store implementation.

When a TransactionLoggable object is written to the persistent store, the class name is persisted and used during recovery to instantiate a new instance of the TransactionLoggable class. The TransactionLoggable allowlist restricts the writing and reading of TransactionLoggable classes to and from the persistent store.

The allowlist is disabled by default. When enabled, the allowlist is populated with a set of WLS-internal TransactionLoggable classes.

- To enable the default allowlist, set the system property to `weblogic.transaction.loggable.allowList`.
- To enable and add classes to the default allowlist, set the system property with a comma-separated list of fully qualified class names. For example, `weblogic.transaction.loggable.allowList=p1.ClassA,p2.ClassB`.

4

Customizing the Default Security Configuration

Oracle WebLogic Server provides a default security configuration that can be customized if you want to replace the default security settings in order to simplify the management of security.

This chapter includes the following sections:

For information about configuring security providers, see About Configuring WebLogic Security Providers and About Configuring the Authentication Providers in WebLogic Server.

For information about migrating security data to a new security realm, see Migrating Security Data.

Why Customize the Default Security Configuration?

In the default security configuration, `myrealm` is set as the default (active) security realm, and the WebLogic Adjudication, Authentication, Identity Assertion, Credential Mapping, CertPath, XACML Authorization and XACML Role Mapping providers are defined as the security providers in the security realm.

Customize the default security configuration if you want to do any of the following:

- Replace one of the security providers in the default realm with a different security provider.
- Configure additional security providers in the default security realm. (For example, if you want to use two Authentication providers.)
- Use an Authentication provider that accesses an LDAP server other than WebLogic Server's embedded LDAP server.
- Use an existing store of users and groups (for example, a DBMS database) instead of defining users and groups in the WebLogic Authentication provider (also known as the DefaultAuthenticator).
- When performing authentication, use the GUID or DN attributes of principals, in addition to user names, specify that principal matching is case-insensitive.
- Add an Auditing provider to the default security realm.
- Use an Identity Assertion provider that handles SAML assertions or Kerberos tokens.
- Use the Certificate Registry to add certificate revocation to the security realm.
- Change the default configuration settings of the security providers.
- Use a custom Authorization or Role Mapping provider that does not support parallel security policy and role modification, respectively, in the security provider database.

For information about configuring different types of security providers in a security realm, see About Configuring WebLogic Security Providers and About Configuring the Authentication Providers in WebLogic Server.

The easiest way to customize the default security configuration is to add the security providers you want to the default security realm (`myrealm`). However, Oracle recommends instead that you customize the default security configuration by creating an entirely new security realm. This preserves your ability to revert more easily to the default security configuration. You configure security providers for the new realm; migrate any security data, such as users as groups, from the existing default realm; and then set the new security realm as the default realm. See Creating and Configuring a New Security Realm: Main Steps.

Before You Create a New Security Realm

Before you create a security realm, determine the set of the security providers you want to use, as well as the model for establishing security policies.

Note the following:

- WebLogic Server includes a wide variety of security providers and, in addition, allows you to create or obtain custom security providers. A valid security realm requires an Authentication provider, an Authorization provider, an Adjudication provider, a Credential Mapping provider, a Role Mapping provider, and a CertPathBuilder. In addition, a security realm can optionally include Identity Assertion, Auditing, and Certificate Registry providers. If your new security realm includes two or more providers of the same type (for example, more than one Authentication provider or more than one Authorization provider), you need to determine how these providers should interact with each other. See Using More Than One Authentication Provider.

  In addition, custom Authorization and Role Mapping providers may or may not support parallel security policy and role modification, respectively, in the security provider database. If your custom Authorization and Role Mapping security providers do not support parallel modification, the WebLogic Security framework can enforce a synchronization mechanism that results in each application and module being placed in a queue and deployed sequentially. To do this, set the **Deployable Provider Synchronization Enabled** and **Deployable Provider Synchronization Timeout** controls for the realm.
- The security roles and policies for Web application and EJB resources can be set through deployment descriptors or through the WebLogic Remote Console. See Options for Securing Web Application and EJB Resources in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*.
- If you are configuring a custom Authorization provider that uses the Web resource (instead of the URL resource) in the new security realm, enable Use Deprecated Web Resource on the new security realm. This option changes the runtime behavior of the Servlet container to use a Web resource rather than a URL resource when performing authorization. Note that the Web resource is deprecated in this release of WebLogic Server.

> **Note:**
>
> When you create a new security realm, you must configure at least one of the Authentication providers to return asserted LoginModules. Otherwise, `run-as` tags defined in deployment descriptors will not work.

See Create a Security Realm in *Oracle WebLogic Remote Console Online Help* .

Creating and Configuring a New Security Realm: Main Steps

The main steps to configure a new security realm include choosing a realm name, selecting and configuring the set of required security providers, creating the appropriate security policies for protecting the WebLogic resources in the realm, and protecting the users accounts that are defined in the realm.

To create a new security realm:

1. Define a name and set the configuration options for the security realm. See Before You Create a New Security Realm and Create a Security Realm in *Oracle WebLogic Remote Console Online Help*.
2. Configure the required security providers for the security realm. A valid security realm requires an Authentication provider, an Authorization provider, an Adjudication provider, a Credential Mapping provider, a Role Mapping provider, and a CertPathBuilder. See About Configuring WebLogic Security Providers and About Configuring the Authentication Providers in WebLogic Server.
3. Optionally, define Identity Assertion, Auditing, and Certificate Registry providers. See About Configuring WebLogic Security Providers and About Configuring the Authentication Providers in WebLogic Server.
4. If you configured the Default Authentication, Authorization, Credential Mapping or Role Mapping provider or the Certificate Registry in the new security realm, verify that the settings of the embedded LDAP server are appropriate. See Managing the Embedded LDAP Server.
5. Optionally, configure caches to improve the performance of the WebLogic or LDAP Authentication providers in the security realm. See Improving the Performance of LDAP Authentication Providers.
6. Protect WebLogic resources in the new security realm with security policies. Creating security policies is a multi-step process with many options. To fully understand this process, read *Securing Resources Using Roles and Policies for Oracle WebLogic Server* in conjunction with this document to ensure security is completely configured for a WebLogic Server deployment.
7. If the security data (users and groups, roles and policies, and credential maps) defined in the existing security realm will also be valid in the new security realm, you can export the security data from the existing realm and import it into the new security realm. See Migrating Security Data.
8. Protect user accounts in the new security realm from dictionary attacks by setting lockout attributes. See Protecting User Accounts.
9. Optionally, set the new realm as the default administrative realm for the WebLogic domain. See Change Default Security Realm in *Oracle WebLogic Remote Console Online Help*.

> **Note:**
>
> You can also use the WebLogic Scripting Tool or Java Management Extensions (JMX) APIs to create a new security configuration. See *Understanding the WebLogic Scripting Tool*.

Using Automatic Realm Restart

WebLogic Server supports the concept of a user-controlled automatic realm restart. Realm restart is the process of initializing a new running instance of a security realm after non-dynamic configuration changes have been made. Realm restart allows non-dynamic configuration changes to take effect immediately without requiring a server restart. WebLogic Server determines if any non-dynamic changes are made to the realm, or to security providers within the realm. When you commit non-dynamic security configuration changes to a realm that do not require a restart, then the realm restart occurs automatically when changes are committed in WebLogic Remote Console.

The Impact of Dynamic and Non-Dynamic Configuration Changes on Realm Restart

The type of configuration change you make determines how realm restart impacts that change. The following three scenarios demonstrate the type of changes you can make to the realm, to a security provider, or another WebLogic configuration, and how automatic realm restart affects those changes:

- Dynamic Changes: Some changes that you make in WebLogic Remote Console take effect immediately when you commit your changes. These changes are called dynamic changes and do not require a server restart or a realm restart.
- Non-dynamic changes to realm or provider that *do not* require a server restart: If you make changes to the non-dynamic attributes of the realm or security provider, and if automatic realm restart has been enabled for that realm, then realm is restarted when your changes are committed. Therefore, a server restart is not required. If automatic realm restart is *not* enabled for a realm, then a server restart is required for non-dynamic changes to that realm or provider.
- Non-dynamic changes to realm or provider and to other WebLogic Server configuration: When non-dynamic configuration changes are made to both the security realm and to other (that is, non-security related) WebLogic domain attributes that *do* require a server restart, the realm is not restarted even if automatic restart is configured and enabled for that realm. In such cases, a server restart is required.

Configuration Options for Realm Restart

If you want to configure automatic realm restart, see Enable Automatic Realm Restart in *Oracle WebLogic Remote Console Online Help*. You can also use the `AutoRestartOnNonDynamicChanges` attribute of the `RealmMBean` to enable or disable automatic restart of the realm if non-dynamic changes are made to the realm or providers within the realm.

A realm restart implies that WebLogic Server initializes a new realm instance with the configuration changes that you made to the previous realm instance. As a result, the old (previous) realm instance is shut down. When the new instance is initialized, the realm object references from the previous instance are migrated to the new instance. However, operations on the old realm instance may still be in progress at the time the new instance is ready. The retire timeout allows in-progress operations to complete without being interrupted while the old instance remains running for the specified timeout period. Use the `RetireTimeoutSeconds` attribute of the RealmMBean to specify the time (in seconds) that you require before the old realm instance shuts down or retires. The minimum value for this attribute is 1 second, whereas the default value is 1 minute (60 seconds).

The default security realm setting provides compatibility with previous WebLogic behavior because a custom security provider may not be able to support realm restart. Therefore, in the

default security realm, automatic realm restart is disabled by default. However, in the new security realms that you create, automatic realm restart is enabled by default.

For more information about `RealmMBean` attributes, see RealmMBean in *MBean Reference for Oracle WebLogic Server*.

Part II
Configuring Security Providers

Security providers are modules that "plug into" an Oracle WebLogic Server security realm to provide security services to applications, such as authentication, authorization, role and credential mapping, auditing, and many more.

This part explains how to configure the security providers provided by WebLogic Server.

> **Note:**
>
> WebLogic Server includes so many Authentication providers and Identity Assertion providers that they are presented in a separate section. See Configuring Authentication Providers.

5

About Configuring WebLogic Security Providers

Although most WebLogic security providers can run with their default settings as soon as Oracle WebLogic Server is started, several providers typically require configuration settings tailored to the environment in which they run. For example, if you are using an identity store other than the embedded LDAP server, you need to configure an Authentication provider that is specific to that store. And if you configure multiple providers of a certain type, you need to specify the order in which they are invoked.
This chapter includes the following sections:

When Do You Need to Configure a Security Provider?

By default, most WebLogic security providers are generally configured to run after you install WebLogic Server. However, the following circumstances require you to supply configuration information:

- Before using the WebLogic Identity Assertion provider, define the active token type. See Configuring Identity Assertion Providers.
- To map tokens to a user in a security realm, configure the user name mapper in the WebLogic Identity Assertion provider. See Configuring a WebLogic Credential Mapping Provider.
- To use auditing in the default (active) security realm, configure either the WebLogic Auditing provider or a custom Auditing provider. See Configuring the WebLogic Auditing Provider.
- To use HTTP and Kerberos-based authentication in conjunction with WebLogic Server. See Configuring Single Sign-On with Microsoft Clients.
- To use identity assertion based on SAML assertions. See Configuring Single Sign-On with Web Browsers and HTTP Clients Using SAML.
- To use certificate revocation. See Configuring the Certificate Lookup and Validation Framework.
- To use an LDAP server other than the embedded LDAP server, configure one of the LDAP Authentication providers. An LDAP authentication provider can be used instead of or in addition to the WebLogic Authentication provider. See Configuring LDAP Authentication Providers.
- To access user, password, group, and group membership information stored in databases for authentication purposes. See Configuring RDBMS Authentication Providers. The RDBMS Authentication providers can be used to upgrade from the RDBMS security realm.
- When you create a new security realm, configure security providers for that realm. See Creating and Configuring a New Security Realm: Main Steps.

- When you add a custom security provider to a security realm or replace a WebLogic security provider with a custom security provider, configure options for the custom security provider.

You can use either the WebLogic-supplied security providers or a custom security provider in a security realm. To configure a custom security provider, see Configure Custom Security Providers in *Oracle WebLogic Remote Console Online Help*.

Reordering Security Providers

You can configure more than one security provider of a given type in a security realm. For example, you might use two or more different Role Mapping providers or Authorization providers. If you have more than one security provider of the same type in a security realm, the order in which these providers are called can affect the overall outcome of the security processes. By default, security providers are called in the order that they were added to the realm. You can use WebLogic Remote Console to change the order of the providers.

Enabling Synchronization in Security Policy and Role Modification at Deployment

For the best performance, and by default, Weblogic Server supports parallel modification to security policy and roles during application and module deployment. For this reason, deployable Authorization and Role Mapping providers configured in the security realm should support parallel calls. The WebLogic deployable XACML Authorization and Role Mapping providers meet this requirement.
However, custom deployable Authorization and Role Mapping providers may or may not support parallel calls. If your custom deployable Authorization or Role Mapping providers do not support parallel calls, you need to disable the parallel security policy and role modification and instead enforce a synchronization mechanism that results in each application and module being placed in a queue and deployed sequentially. Otherwise, if a provider does not support parallel calls, it generates a `java.util.ConcurrentModificationException` exception.

You can turn on this synchronization enforcement mechanism on in two ways:

> **Note:**
>
> Enabling the synchronization mechanism affects every deployable provider configured in the realm, including the WebLogic Server XACML providers. Enabling the synchronization mechanism may negatively impact the performance of these providers.

- In WebLogic Remote Console, in the **Edit Tree**, go to **Security**, then **Realms**, then *myRealm*. Click **Show Advanced Fields**. Set the **Deployable Provider Synchronization Enabled** and **Deployable Provider Synchronization Timeout** options for the realm.

  The **Deployable Provider Synchronization Enabled** option enforces a synchronization mechanism that results in each application and module being placed in a queue and deployed sequentially.

  The **Deployable Provider Synchronization Timeout** option sets or returns the timeout value, in milliseconds, for the deployable security provider synchronization operation. This is the maximum time a deployment cycle wants to wait in the queue when the previous cycle is stuck.

- From WLST, set the `DeployableProviderSynchronizationEnabled` and `DeployableProviderSynchronizationTimeout` attributes of the RealmMBean.

  See RealmMBean in *MBean Reference for Oracle WebLogic Server*.

6

Configuring Authorization and Role Mapping Providers

In Oracle WebLogic Server, Authorization providers use the concepts of security policies, ContextHandlers, access decisions, and more, to determine who may have access to a resource. Role Mapping providers compute the set of roles granted to a subject for a given resource, and Adjudication providers resolve authorization conflicts if multiple Authorization providers don't return the same access decision.
This chapter includes the following sections:

Configuring an Authorization Provider

Authorization is the process whereby the interactions between users and resources are limited to ensure integrity, confidentiality, and availability. In other words, authorization is responsible for controlling access to resources based on user identity or other information. You should only need to configure an Authorization provider when you create a new security realm.

By default, security realms in newly created domains include the XACML Authorization provider. The XACML Authorization provider uses XACML, the eXtensible Access Control Markup Language. For information about using the XACML Authorization provider, see Using XACML Documents to Secure WebLogic Resources in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*. WebLogic Server also includes the WebLogic Authorization provider, which uses a proprietary policy language. This provider is named DefaultAuthorizer, but is no longer the default authorization provider.

See Enabling Synchronization in Security Policy and Role Modification at Deployment for information about how Authorization providers support parallel modification to security policy during application and module deployment.

> **Note:**
>
> The WebLogic Authorization provider, also known as the DefaultAuthorizer, is deprecated in WebLogic Server 14.1.1.0.0 and will be removed in a future release.
>
> The WebLogic Authorization provider improves performance by caching the roles, predicates, and resource data that it looks up. For information on configuring these caches, see Best Practices: Configure Entitlements Caching When Using WebLogic Providers in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*. The XACML Authorization uses its own cache, but this cache is not configurable.

Configuring the WebLogic Adjudication Provider

When multiple Authorization providers are configured in a security realm, each may return a different answer to the "is access allowed" question for a given resource. This answer may be `PERMIT`, `DENY`, or `ABSTAIN`. Determining what to do if multiple Authorization providers do not agree on the answer is the primary function of the Adjudication provider. Adjudication providers resolve authorization conflicts by weighting each Authorization provider's answer and returning a final decision.
Each security realm requires an Adjudication provider, and can have no more than one active Adjudication provider. By default, a WebLogic security realm is configured with the WebLogic Adjudication provider. You can use either the WebLogic Adjudication provider or a custom Adjudication provider in a security realm.

> **Note:**
>
> In the WebLogic Remote Console, the WebLogic Adjudication provider is referred to as the Default Adjudicator.

By default, most configuration options for the WebLogic Adjudication provider are defined. However, you can set the Require Unanimous Permit option to determine how the WebLogic Adjudication provider handles a combination of `PERMIT` and `ABSTAIN` votes from the configured Authorization providers.

- If the option is enabled (the default), all Authorization providers must vote `PERMIT` in order for the Adjudication provider to vote `true`.
- If the option is disabled, `ABSTAIN` votes are counted as `PERMIT` votes.

Configuring a Role Mapping Provider

Role mapping is the process whereby principals (users or groups) are dynamically mapped to security roles at runtime. Role Mapping providers supply Authorization providers with this role information so that the Authorization provider can answer the "is access allowed?" question for WebLogic resources. By default, a WebLogic security realm is configured with the XACML Role Mapping provider. The XACML Role Mapping provider uses XACML, the eXtensible Access Control Markup Language. For information about using the XACML Role Mapping provider, see Using XACML Documents to Secure WebLogic Resources in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*.
WebLogic Server also includes the WebLogic Role Mapping provider, which uses a proprietary policy language. This provider is named DefaultRoleMapper, but is no longer the default role mapping provider in newly-created security realms. You can also use a custom Role Mapping provider in your security realm.

> **Note:**
>
> The WebLogic Role Mapping provider, also known as the DefaultRoleMapper, is deprecated in WebLogic Server 14.1.1.0.0 and will be removed in a future release.

By default, most configuration options for the XACML Role Mapping provider are already defined. However, you can set Role Mapping Deployment Enabled, which specifies whether or not this Role Mapping provider imports information from deployment descriptors for Web applications and EJBs into the security realm. This setting is enabled by default.

In order to support Role Mapping Deployment Enabled, a Role Mapping provider must implement the `DeployableRoleProvider` SSPI. Roles are stored by the XACML Role Mapping provider in the embedded LDAP server.

See Enabling Synchronization in Security Policy and Role Modification at Deployment for information about how Role Mapping providers support parallel modification to roles during application and module deployment.

For information about using, developing, and configuring Role Mapping providers, see:

- Users, Groups, And Security Roles in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*
- Role Mapping Providers in *Developing Security Providers for Oracle WebLogic Server*
- Configure a Role Mapping Provider in *Oracle WebLogic Remote Console Online Help*

> **Note:**
>
> The WebLogic Role Mapping provider improves performance by caching the roles, predicates, and resource data that it looks up. For information on configuring these caches, see Best Practices: Configure Entitlements Caching When Using WebLogic Providers in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*. The XACML Role Mapping provider uses its own cache, but this cache is not configurable.

7

Configuring the WebLogic Auditing Provider

The Oracle WebLogic Server WebLogic Security Framework invokes an Auditing provider before and after security operations, such as authentication or authorization, have been performed, when changes to the domain configuration are made, or when management operations on any resources in the domain are invoked. The decision to audit a particular event is made by the Auditing provider itself and can be based on specific audit criteria or severity levels. The records containing the audit information may be written to output repositories such as an LDAP server, database, or a simple file.
This chapter contains the following sections:

Auditing Provider Overview

Auditing is the process whereby information about operating requests and the outcome of those requests are collected, stored, and distributed for the purposes of non-repudiation. In other words, Auditing providers produce an electronic trail of computer activity.
Configuring an Auditing provider is optional. The default security realm (`myrealm`) does not have an Auditing provider configured. WebLogic Server includes a provider named the WebLogic Auditing provider (referred to as `DefaultAuditor` in the WebLogic Remote Console). You can also develop custom Auditing providers, as described in Auditing Providers in *Developing Security Providers for Oracle WebLogic Server*.

Events Logged by the WebLogic Auditing Provider

If the WebLogic Auditing Provider is enabled, then it can log events such as authentication, authorization, user account status, and more. The WebLogic Auditing provider can log the events described in Table 7-1.

Table 7-1 WebLogic Auditing Provider Events

The following audit event . . .	Indicates . . .
`AUTHENTICATE`	A simple authentication (username and password) occurred.
`ASSERTIDENTITY`	A perimeter authentication (based on tokens) occurred.
`USERLOCKED`	A user account is locked because of invalid login attempts.
`USERUNLOCKED`	The lock on a user account is cleared.

Table 7-1 (Cont.) WebLogic Auditing Provider Events

The following audit event . . .	Indicates . . .
USERLOCKOUTEXPIRED	The lock on a user account expired.
ISAUTHORIZED	An authorization attempt occurred.
ROLEEVENT	A getRoles event occurred.
ROLEDEPLOY	A deployRole event occurred.
ROLEUNDEPLOY	An undeployRole event occurred.
POLICYDEPLOY	A deployPolicy event occurred.
POLICYUNDEPLOY	An undeployPolicy event occurred.
START_AUDIT	An Auditing provider has been started.
STOP_AUDIT	An Auditing provider has been stopped.

Configuration Options

By default, most configuration options for the WebLogic Auditing provider are already defined and, once it is added to the active security realm, the WebLogic Auditing provider will begin to record audit events. However, you need to define the following settings, which you can do in the WebLogic Remote Console under the **Default Auditor Parameters** tab of the Auditing provider page. You can also use WebLogic Scripting tool or the Java Management Extensions (JMX) APIs to configure the Auditing provider:

- Rotation Type - Specifies the criteria for moving old audit records to a new file. Can be either byTime or bySize.
- Rotation Minutes—Specifies how many minutes to wait before creating a new DefaultAuditRecorder.log file. At the specified time, the audit file is closed and a new one is created. A backup file named DefaultAuditRecorder.*YYYYMMDDHHMM*.log (for example, DefaultAuditRecorder.200405130110.log) is created in the same directory. Rotation Size must be set to byTime.
- Rotation Size - Specifies the file size to reach before creating a new audit log file. Rotation Size must be set to bySize.
- Severity—Severity level appropriate for your WebLogic Server deployment. The WebLogic Auditing provider audits security events of the specified severity, as well as all events with a higher numeric severity rank. For example, if you set the severity level to ERROR, the WebLogic Auditing provider audits security events of severity level ERROR, SUCCESS, and FAILURE. You can also set the severity level to CUSTOM, and then enable the specific severity levels you want to audit, such as ERROR and FAILURE events only. Audit events include both the severity name and numeric rank; therefore, a custom Auditing provider can filter events by either the name or the numeric rank. Auditing can be initiated when the following levels of security events occur.

Table 7-2 Event Severity

Event Severity	Rank
INFORMATION	1
WARNING	2

Table 7-2 (Cont.) Event Severity

Event Severity	Rank
ERROR	3
SUCCESS	4
FAILURE	5

- Number of Files Limit - Specifies the number of audit log files that will be retained.

All auditing information recorded by the WebLogic Auditing provider is saved in *WL_HOME*\yourdomain\yourserver\logs\DefaultAuditRecorder.log by default. Although an Auditing provider is configured per security realm, each server writes auditing data to its own log file in the server directory. You can specify a new directory location for the DefaultAuditRecorder.log file on the command line with the following Java startup option:

```
-Dweblogic.security.audit.auditLogDir=c:\foo
```

The new file location will be c:\foo\yourserver\logs\DefaultAuditRecorder.log.

See Security in the *Command Reference for Oracle WebLogic Server*.

> **Note:**
>
> Using an Auditing provider affects the performance of WebLogic Server even if only a few events are logged.

See Configure an Auditing Provider in *Oracle WebLogic Remote Console Online Help*.

Auditing ContextHandler Elements

The ContextHandler interface is used to manage audit providers that support context handler entries. Set the ContextHandler Entries attribute to specify which ContextElement entries are recorded by the Auditing provider.

An Audit Event includes a ContextHandler that can hold a variety of information or objects. Set the WebLogic Auditing provider's Active ContextHandler Entries attribute to specify which ContextElement entries in the ContextHandler are recorded by the Auditing provider. By default, none of the ContextElements are audited. Objects in the ContextHandler are in most cases logged using the toString method. Table 7-3 lists the available ContextHandler entries.

> **Note:**
>
> The WebLogic Auditing provider can audit only the attributes for the specific functionality that is being implemented. It does not audit all of the context handler elements by default. For example, if you log into the WebLogic Remote Console using HTTP, the authentication is performed in the context of an HTTP servlet request, and the Auditing provider audits HTTP servlet elements. Alternatively, authentication from WLST uses t3 protocol. For t3 authentication, the auditing provider audits the channel context elements such as `com.bea.contextelement.channel.Protocol` and `com.bea.contextelement.channel.RemoteAddress`. In both cases, the Auditing provider only audits the functionality being implemented, either HTTP or t3.

Table 7-3 Context Handler Entries for Auditing

Context Element Name	Description and Type
`com.bea.contextelement.servlet.HttpServletRequest`	A servlet access request or SOAP message via HTTP `javax.http.servlet.HttpServletRequest`
`com.bea.contextelement.servlet.HttpServletResponse`	A servlet access response or SOAP message via HTTP `javax.http.servlet.HttpServletResponse`
`com.bea.contextelement.wli.Message`	An Oracle WebLogic Integration message. The message is streamed to the audit log. `java.io.InputStream`
`com.bea.contextelement.channel.Port`	Internal listen port of the network channel accepting or processing the request `java.lang.Integer`
`com.bea.contextelement.channel.PublicPort`	External listen port of the network channel accepting or processing the request `java.lang.Integer`
`com.bea.contextelement.channel.RemotePort`	Port of the remote end of the TCP/IP connection of the network channel accepting or processing the request `java.lang.Integer`
`com.bea.contextelement.channel.Protocol`	Protocol used to make the request of the network channel accepting or processing the request `java.lang.String`
`com.bea.contextelement.channel.Address`	The internal listen address of the network channel accepting or processing the request `java.lang.String`
`com.bea.contextelement.channel.PublicAddress`	The external listen address of the network channel accepting or processing the request `java.lang.String`
`com.bea.contextelement.channel.RemoteAddress`	Remote address of the TCP/IP connection of the network channel accepting or processing the request `java.lang.String`
`com.bea.contextelement.channel.ChannelName`	Name of the network channel accepting or processing the request `java.lang.String`

Table 7-3 (Cont.) Context Handler Entries for Auditing

Context Element Name	Description and Type
com.bea.contextelement.channel.Secure	Whether the network channel is accepting or processing the request using SSL java.lang.Boolean
com.bea.contextelement.ejb20.Parameter[1-N]	Object based on parameter
com.bea.contextelement.wsee.SOAPMessage	javax.xml.rpc.handler.MessageContext
com.bea.contextelement.entitlement.EAuxiliaryID	Used by a WebLogic Server internal process. weblogic.entitlement.expression.EAuxiliary
com.bea.contextelement.security.ChainPrevalidatedBySSL	SSL framework has validated the certificate chain, meaning that the certificates in the chain have signed each other properly; the chain terminates in a certificate that is one of the server's trusted CAs; the chain honors the basic constraints rules; and the certificates in the chain have not expired. java.lang.Boolean
com.bea.contextelement.xml.SecurityToken	Not used in this release of WebLogic Server. weblogic.xml.crypto.wss.provider.SecurityToken
com.bea.contextelement.xml.SecurityTokenAssertion	Not used in this release of WebLogic Server. java.util.Map
com.bea.contextelement.webservice.Integrity{id:XXXXX}	javax.security.auth.Subject
com.bea.contextelement.saml.SSLClientCertificateChain	SSL client certificate chain obtained from the SSL connection over which a sender-vouches SAML assertion was received. java.security.cert.X509Certificate[]
com.bea.contextelement.saml.MessageSignerCertificate	Certificate used to sign a Web services message. java.security.cert.X509Certificate
com.bea.contextelement.saml.subject.ConfirmationMethod	Type of SAML assertion: bearer, artifact, sender-vouches, or holder-of-key. java.lang.String
com.bea.contextelement.saml.subject.dom.KeyInfo	<ds:KeyInfo> element to be used for subject confirmation with holder-of-key SAML assertions. org.w3c.dom.Element

Configuration Auditing

You can configure the Administration Server to emit log messages and generate audit events when a user changes the configuration of any resource within a domain or invokes management operations on any resource within a domain. For example, if a user disables SSL on a Managed Server in a domain, the Administration Server emits log messages. If you have enabled the WebLogic Auditing provider, it writes the audit events to an additional security log. These messages and audit events provide an audit trail of changes within a domain's configuration (configuration auditing).

The Administration Server writes configuration auditing messages to its local log file. They are not written to the domain-wide message log by default.

Note that configuration audit information is contained in Authorization Events. As a result, another approach to configuration auditing is to consume Authorization Events. Note, however, that the information in an Authorization Event tells you whether access was allowed to perform a configuration change; it does not tell you whether the configuration change actually succeeded (for instance, it might have failed because it was invalid).

To enable configuration auditing, see Enabling Configuration Auditing.

Enabling Configuration Auditing

Learn the different methods to enable configuration auditing.
Enable configuration auditing by one of these methods:

- Use WebLogic Remote Console. See Enable Configuration Auditing in *Oracle WebLogic Remote Console Online Help*.
- When you start the Administration Server, include one of the following Java options in the `weblogic.Server` command:
  - `-Dweblogic.domain.ConfigurationAuditType="audit"`

    Causes the domain to emit Audit Events only.
  - `-Dweblogic.domain.ConfigurationAuditType="log"`

    Causes the domain to write configuration auditing messages to the Administration Server log file only.
  - `-Dweblogic.domain.ConfigurationAuditType="logaudit"`

    Causes the domain to emit Audit Events and write them to the auditor's log file, while log events are written to the Administration Server log file. If you are using a custom auditing provider, then the custom auditing provider determines where the Audit Events are written to.

  See weblogic.Server Command-Line Reference in *Command Reference for Oracle WebLogic Server*.
- Use the WebLogic Scripting Tool to change the value of the `ConfigurationAuditType` attribute of the `DomainMBean`. See *Understanding the WebLogic Scripting Tool*.

Configuration Auditing Messages

The configuration auditing severity levels are `SUCCESS`, `FAILURE`, and `ERROR`.

Table 7-4 Configuration Auditing Message Severities

Severity	Description
SUCCESS	A successful configuration change occurred.
FAILURE	An attempt to modify the configuration failed due to insufficient user credentials.
ERROR	An attempt to modify the configuration failed due to an internal error.

Configuration auditing messages are identified by message IDs that fall within the range of `159900-159910`.

The messages use MBean object names to identify resources. Object names for WebLogic Server MBeans reflect the location of the MBean within the hierarchical data model. To reflect the location, object names contain name/value pairs from the parent MBean. For example, the object name for a server's `LogMBean` is:
`mydomain:Name=myserverlog,Type=Log,Server=myserver`. See WebLogic Server MBean Data Model in *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*.

Table 7-5 summarizes the messages.

Table 7-5 Summary of Configuration Auditing Messages

When this event occurs . . .	WebLogic Server generates a message with this ID . . .	And this message text . . .
Authorized user creates a resource.	159900	`USER username CREATED MBean-name` where *username* identifies the WebLogic Server user who logged in and created a resource.
Unauthorized user attempts to create a resource.	159901	`USER username CREATED MBean-name FAILED weblogic.management. NoAccessRuntimeException: exception-text stack-trace` where *username* identifies the unauthorized WebLogic Server user.
Authorized user deletes a resource.	159902	`USER username REMOVED MBean-name` where *username* identifies the WebLogic Server user who logged in and deleted a resource.
Unauthorized user attempts to delete a resource.	159903	`USER username REMOVE MBean-name FAILED weblogic.management. NoAccessRuntimeException: exception-text stack-trace` where *username* identifies the unauthorized WebLogic Server user.
Authorized user changes a resource's configuration.	159904	`USER username MODIFIED MBean-name ATTRIBUTE attribute-name FROM old-value TO new-value` where *username* identifies the WebLogic Server user who logged in and changed the resource's configuration.
Unauthorized user attempts to change a resource's configuration.	159905	`USER username MODIFY MBean-name ATTRIBUTE attribute-name FROM old-value TO new-value FAILED weblogic.management. NoAccessRuntimeException: exception-text stack-trace` where *username* identifies the unauthorized WebLogic Server user.
Authorized user invokes an operation on a resource. For example, a user deploys an application or starts a server instance.	159907	`USER username INVOKED ON MBean-name METHOD operation-name PARAMS specified-parameters` where *username* identifies the WebLogic Server user who logged in and invoked a resource operation.

Table 7-5 (Cont.) Summary of Configuration Auditing Messages

When this event occurs . . .	WebLogic Server generates a message with this ID . . .	And this message text . . .
Unauthorized user attempts to invoke an operation on a resource.	159908	`USER username INVOKED ON MBean-name METHOD operation-name PARAMS specified-parameters FAILED weblogic.management. NoAccessRuntimeException: exception-text stack-trace` where *username* identifies the unauthorized WebLogic Server user.
Authorized user enables configuration auditing.	159909	`USER username, Configuration Auditing is enabled` where *username* identifies the WebLogic Server user who enabled configuration auditing.
Authorized user disables configuration auditing.	159910	`USER username, Configuration Auditing is disabled` where *username* identifies the WebLogic Server user who disabled configuration auditing.

Note:

Each time an authorized user adds, modifies, or deletes a resource, the Management subsystem also generates an Info message with the ID 140009 regardless of whether configuration auditing is enabled. For example:

```
<Sep 15, 2005 11:54:47 AM EDT> <Info> <Management> <140009>
<Configuration changes for domain saved to the repository.>
```

While the message informs you that the domain's configuration has changed, it does not provide the detailed information that configuration auditing messages provide. Nor does the Management subsystem generate this message when you invoke operations on resources.

Table 7-6 lists additional message attributes for configuration auditing messages. All configuration auditing messages specify the same values for these attributes.

Table 7-6 Common Message Attributes and Values

Message Attribute	Attribute Value
Severity	`Info`
Subsystem	`Configuration Audit`
User ID	`kernel identity` This value is always `kernel` identity, regardless of which user modified the resource or invoked the resource operation.

Table 7-6 (Cont.) Common Message Attributes and Values

Message Attribute	Attribute Value
Server Name	*AdminServerName* Because the Administration Server maintains the configuration data for all resources in a domain, this value is always the name of the Administration Server.
Machine Name	*AdminServerHostName* Because the Administration Server maintains the configuration data for all resources in a domain, this value is always the name of the Administration Server's host machine.
Thread ID	*execute-thread* The value depends on the number of execute threads that are currently running on the Administration Server.
Timestamp	*timeStamp* at which the message is generated.

Audit Events and Auditing Providers

An audit event is an object that Auditing providers can read and process in specific ways. An Auditing provider is a pluggable component that the security realm uses to collect, store, and distribute information about operating requests and the outcome of those requests for the purposes of non-repudiation.
If you enable a domain to emit Audit Events, the domain emits the events described in Table 7-7. All Auditing providers that are configured for the domain can handle these events.

All of the events are of severity level `SUCCESS` and describe the security principal who initiated the action, whether permission was granted, and the object (MBean or MBean attribute) of the requested action.

Table 7-7 Summary of Audit Events for Configuration Auditing

When this event occurs . . .	WebLogic Server generates this Audit Event object . . .
A request to create a new configuration artifact has been allowed or prevented.	weblogic.security.spi.AuditCreateConfigurationEvent
A request to delete an existing configuration artifact has been allowed or prevented.	weblogic.security.spi.AuditDeleteConfigurationEvent
A request to modify an existing configuration artifact has been allowed or prevented.	weblogic.security.spi.AuditInvokeConfigurationEvent
A invoke an operation on an existing configuration artifact has been allowed or prevented.	weblogic.security.spi.AuditSetAttributeConfigurationEvent

If you enable the default WebLogic Server Auditing provider, it writes all Audit Events as log messages in its own log file.

Other Auditing providers that you create or purchase can filter these events and write them to output repositories such as an LDAP server, database, or a simple file. In addition, other types

of security providers can request audit services from an Auditing provider. See Auditing Providers in *Developing Security Providers for Oracle WebLogic Server*.

8

Configuring Credential Mapping Providers

Credential mapping is the process whereby the authentication and authorization mechanisms of a remote system (for example, a legacy system or application) obtain an appropriate set of credentials to authenticate remote users to a target WebLogic resource. The WebLogic Credential Mapping provider maps Oracle WebLogic Server subjects to the username/password pairs to be used when accessing such resources.
The following topics are included:

Configuring a WebLogic Credential Mapping Provider

By default, most configuration options for the WebLogic Credential Mapping provider are defined. You do have the option of adjusting the expiration interval of the `weblogic-jwt-token` token type, which is used internally to propagate identity for REST invocations of other applications running in the domain. By default, the expiration interval is set to 3 minutes. However, you can adjust the interval from the Provider Specific configuration page for this security provider.

> **Note:**
>
> WebLogic Server provides the option of setting Credential Mapping Deployment Enabled, which specifies whether or not the Credential Mapping provider imports credential maps from a resource adapter's deployment descriptor (`weblogic-ra.xml` file) into the security realm. However, this option is now deprecated. Deploying credential maps from a `weblogic-ra.xml` file is no longer supported by WebLogic Server.

In order to support Credential Mapping Deployment Enabled, a Credential Mapping provider must implement the `DeployableCredentialProvider` SSPI. The credential mapping information is stored in the embedded LDAP server.

Refer to the following topics:

- See Credential Mapping Providers in *Developing Security Providers for Oracle WebLogic Server*.
- See Configure a Credential Mapping Provider in *Oracle WebLogic Remote Console Online Help*
- For information about using credential maps, see *Developing Resource Adapters for Oracle WebLogic Server*.

- You can also use the WebLogic Scripting Tool or Java Management Extensions (JMX) APIs to create a new security configuration.

Configuring a PKI Credential Mapping Provider

The Public Key Infrastructure (PKI) Credential Mapping provider included in WebLogic Server maps a WebLogic Server subject and target resource to a key pair or a public certificate that can be used by applications when accessing the targeted resource. The PKI Credential Mapping provider uses the subject and resource name to retrieve the corresponding credential from the keystore.
To use the PKI Credential Mapping provider, you need to:

1. Configure keystores with appropriate keys and distribute the keystores on all machines in a WebLogic Server cluster. Setting up keystores is not a WebLogic Server function. For information about setting up keystores, see the help for the Java keytool utility at the following locations:
   - **Java SE 17** - keytool in *Java Development Kit Version 17 Tool Specifications*
   - **Java SE 21** - keytool in *Java Development Kit Version 21 Tool Specifications*

   See also Configuring Keystores for information about keystores and keys in WebLogic Server.
2. Configure a PKI Credential Mapping provider. A PKI Credential Mapping provider is not already configured in the default security realm (`myrealm`). See PKI Credential Mapper Attributes and Configure a Credential Mapping Provider in *Oracle WebLogic Remote Console Online Help* .
3. Create credential mappings.

This section contains the following topics:

- PKI Credential Mapper Attributes
- Credential Actions

PKI Credential Mapper Attributes

To configure the PKI Credential Mapping provider, set values for these attributes. See Configure a Credential Mapping Provider in *Oracle WebLogic Remote Console Online Help* .

- Keystore Provider—A keystore provider for the Java Security API. If no value is specified, the default provider class is used.
- Keystore Type— JKS (the default) or PKCS12.
- Keystore Pass Phrase—Password used to access the keystore
- Keystore File Name—Location of the keystore relative to the directory where the server was started.

In addition, two optional attributes determine how the PKI Credential Mapping provider locates credential mappings in cases where the exact resource or subject may not be available:

- Use Resource Hierarchy—A credential is located by traversing up the resource hierarchy for each type of resource. The search for all possible PKI credentials will start from the specific resource and will walk up the resource hierarchy to find all possible matches. This attribute is enabled by default.

- Use Initiator Group Names—When a subject is passed to the PKI Credential Mapper provider, a credential is located by examining the groups of which the initiator is a member. This is enabled by default.

Credential Actions

Optionally, you can label a credential mapping with a credential action. You can do this in WebLogic Remote Console when you create the credential mapping. The credential action is an arbitrary string that distinguishes credential mappings used in different circumstances. For example, one credential mapping could decrypt a message from a remote resource and another credential mapping could sign messages to be sent to the same resource. The subject initiator and the target resource are not sufficient to distinguish these two credential mappings. You can use the credential action to label one of these credential mappings something like `decrypt` and the other one `sign`. Then, the container calling the PKI Credential Mapping provider can provide the desired credential action value in the `ContextHandler` that is passed to the provider.

Configuring a SAML Credential Mapping Provider for SAML 1.1

WebLogic Server includes SAML Credential Mapping provider Version 2. It provides greatly enhanced configuration options and is recommended for new deployments. A security realm can have not more than one SAML Credential Mapping provider, and if the security realm has both SAML Credential Mapping provider and a SAML Identity Assertion provider, both must be of the same version.

> **Note:**
>
> The SAML 1.1 credential mapping provider and related configuration and services for SAML 1.1 federation services are deprecated as of WebLogic Server 14.1.2.0.0 and will be removed in a future release. Oracle recommends using SAML 2.0.

For general information about WebLogic Server support for SAML, see Security Assertion Markup Language (SAML) and Single Sign-On with the WebLogic Security Framework in *Understanding Security for Oracle WebLogic Server*. For information about how to use the SAML Credential Mapping provider in a SAML single sign-on configuration, see Configuring Single Sign-On with Web Browsers and HTTP Clients Using SAML.

This section includes the following topics:

- Configuring Assertion Lifetime
- Relying Party Registry

Configuring Assertion Lifetime

A SAML Assertion's validity is typically time-limited. The default time-to-live for assertions generated by the SAML Credential Mapping provider is specified by the `DefaultTimeToLive` attribute. You can override the default time-to-live for assertions generated for different SAML Relying Parties.

Normally, an assertion is valid from the `NotBefore` time, which defaults to (roughly) the time the assertion was generated, until the `NotOnOrAfter` time, which is calculated as (`NotBefore` + `TimeToLive`). To allow the Credential Mapper to compensate for clock differences between the

source and destination sites, you can configure the SAML Credential Mapping provider's `DefaultTimeToLiveDelta` attribute. This time-to-live offset value is a positive or negative integer indicating how many seconds before or after "now" the assertion's `NotBefore` value should be set to. If you set a value for `DefaultTimeToLiveDelta`, then the assertion lifetime is still calculated as (`NotBefore` + `TimeToLive`), but the `NotBefore` value is set to (`now` + `TimeToLiveDelta`). For example, given the following settings:

```
DefaultTimeToLive = 120
DefaultTimeToLiveDelta = -30
```

an assertion when generated would have a lifetime of two minutes (120 seconds), starting 30 seconds before it is generated.

Relying Party Registry

When you configure WebLogic Server to act as a source of SAML security assertions, you need to register the parties that may request SAML assertions to be generated. For each SAML Relying Party, you can specify the SAML profile used, details about the Relying Party, and the attributes expected in assertions for the Relying Party. See Configure Relying Parties.

Configuring a SAML 2.0 Credential Mapping Provider for SAML 2.0

The SAML 2.0 Credential Mapping provider in WebLogic Server generates SAML 2.0 assertions that can be used to assert identity in the SAML 2.0 Web SSO Profile and the WS-Security SAML Token Profile version 1.1 use cases. The SAML 2.0 Credential Mapping provider generates the assertion types listed and described in Table 8-1.

Table 8-1 Assertion Types Supported by the SAML 2.0 Credential Mapping Provider

Assertion Type	Description
`bearer`	The subject of the assertion is the bearer of the assertion, subject to optional constraints on confirmation using attributes that may be included in the `<SubjectConfirmationData>` element of the assertion. Used for all assertions generated for the SAML 2.0 Web Browser SSO Profile and with the Web Service Security SAML Token Profile 1.1.
`sender-vouches`	The Identity Provider (different from the subject) vouches for the verification of the subject. The receiver must have a trust relationship with the Identity Provider. Used with the Web Service Security SAML Token Profile 1.1 only.
`holder-of-key`	The subject represented in the assertion uses an X.509 certificate that may not be trusted by the receiver to protect the integrity of the request messages. Used with the Web Service Security SAML Token Profile 1.1 only.

For general information about WebLogic Server's support for SAML 2.0, see Security Assertion Markup Language (SAML) and Single Sign-On with the WebLogic Security Framework in *Understanding Security for Oracle WebLogic Server*. For information about how to use the SAML 2.0 Credential Mapping provider in a SAML 2.0 single sign-on configuration, see Configuring Single Sign-On with Web Browsers and HTTP Clients Using SAML. For information about specifying the confirmation method for assertions generated for web service Service provider partners, see Using Security Assertion Markup Language (SAML) Tokens For Identity in *Securing WebLogic Web Services for Oracle WebLogic Server*.

This section includes the following topics:

SAML 2.0 Credential Mapping Provider Attributes

Configuration of the SAML 2.0 Credential Mapping provider is controlled by setting attributes on the `SAML2CredentialMapperMBean`. You can access the `SAML2CredentialMapperMBean` using the WebLogic Scripting Tool (WLST), or through WebLogic Remote Console by going to the **Edit Tree**, then **Security**, then **Realms**, then *myRealm* and creating a SAML2CredentialMapper. For details about these attributes, see the description of the SAML2CredentialMapperMBean in the *MBean Reference for Oracle WebLogic Server*.

To configure the SAML 2.0 Credential Mapping provider, set the following attributes:

- **Issuer URI**

  Name of this security provider. The value that you specify should match the Entity ID specified in the SAML 2.0 General page that configures the per-server SAML 2.0 properties.

- **Name Qualifier**

  Used by the Name Mapper class as the security or administrative domain that qualifies the name of the subject. This provides a means to federate names from disparate user stores while avoiding the possibility of subject name collision.

- **Default Time to Live life time**

  Values that limit the life time of generated assertions during which they may be used. Expired assertions cannot be made available for use.

- **Signing Key Alias** and **Signing Key Pass Phrase**

  Used for signing generated assertions.

- **Custom name mapper class**

  The custom Java class that overrides the default SAML 2.0 Credential Mapping provider name mapper class, which maps Subjects to identity information contained in the assertion.

- **Generate attributes**

  Specifies whether group membership information associated with the authenticated Subject is included in generated assertions.

Service Provider Partners

When you configure WebLogic Server to act as an Identity Provider, you need to create and configure the Service Provider partners for whom SAML 2.0 assertions are generated. When an Identity Provider site needs to generate an assertion, the SAML 2.0 Credential Mapping provider determines the Service Provider partner for whom the assertion must be generated, and generates it according to the configuration of that Service Provider partner.

The way in which you configure a Service Provider partner, and the specific information you configure for that partner, depends upon whether the partner is used for web single sign-on or web services. Configuring a web single sign-on Service Provider partner consists of importing that partner's metadata file and establishing additional basic information about that partner, such as the following:

- Determining whether SAML documents, such as authentication responses, SAML artifacts, and artifact requests, must be signed
- Certificates used for validating signed documents received from this partner
- The binding to be used for sending SAML artifacts to this partner
- The client user name and password used by this partner when connecting to the local site's binding

For details about configuring a Service Provider partner for web single sign-on, see:

- Create and Configure Web Single Sign-On Service Provider Partners
- Create a SAML 2.0 Web Single Sign-On Service Provider Partner in *Oracle WebLogic Remote Console Online Help*

Configuring a Web service Service Provider partner does not use a metadata file, but does consist of establishing the following information about that partner:

- Audience URIs, which specify an audience restriction to be included in assertions generated for this partner

  In WebLogic Server, the Audience URI attribute is overloaded to also include the partner lookup string, which is required by the web service run time to discover the partner. See Partner Lookup Strings Required for Web Service Partners.
- Custom name mapper class that overrides the default name mapper and that is to be used specifically with this partner
- Values that specify the life span attributes of assertions generated for this partner
- Confirmation method for assertions received from this partner

For more information about configuring web service Service Provider partners, see Create a SAML 2.0 Web Service Service Provider Partner in *Oracle WebLogic Remote Console Online Help*.

This section includes the following topics:

- Partner Lookup Strings Required for Web Service Partners
- Management of Partner Certificates

Partner Lookup Strings Required for Web Service Partners

For web service Service Provider partners, you also configure Audience URIs. In WebLogic Server, the Audience URI attribute is overloaded to perform two distinct functions:

- Specify an audience restriction that consists of the target service URL, per the OASIS SAML 2.0 specification.
- Contain a partner lookup string, which is required at run time by WebLogic Server to discover the Service Provider partner for which a SAML 2.0 assertion needs to be generated.

The partner lookup string specifies an endpoint URL, which is used for partner lookup and can optionally also serve as an Audience URI restriction that is included in the generated assertion. The ability to specify a partner lookup string that is also an Audience URI eliminates the need to specify a given target URL twice: once for lookup, and again for audience restriction.

> **Note:**
>
> You must configure a partner lookup string for a Service Provider partner so that partner can be discovered at run time by the web service run time.

This section includes the following topics:

- Lookup String Syntax
- Specifying Default Partners

Lookup String Syntax

The partner lookup string has the following syntax:

```
[target:char:]<endpoint-url>
```

In this syntax, `target:char:` is a prefix that designates the partner lookup string, where *`char`* represents one of three special characters: a hyphen, plus sign, or asterisk (-, +, or *). This prefix determines how partner lookup is performed, as described in Table 8-2.

Table 8-2 Service Provider Partner Lookup String Syntax

Lookup String	Description
`target:-:<endpoint-url>`	Specifies that partner lookup is conducted for an exact match of the URL, `<endpoint-url>`. For example, `target:-:http://www.avitek.com:7001/myserver/myservicecontext/myservice-endpoint` specifies the endpoint that can be matched to this Service Provider, for which an assertion should be generated. This form of partner lookup string excludes the endpoint URL from being added as an Audience URI in the generated assertion.
`target:+:<endpoint-url>`	Specifies that partner lookup is conducted for an exact match of the URL, `<endpoint-url>`. Using the plus sign (+) in the lookup string results in the endpoint URL being added as an Audience URI in the assertion generated for this Service Provider partner.
`target:*:<endpoint-url>`	Specifies that partner lookup is conducted for an initial-string pattern match of the URL, `<endpoint-url>`. For example, `target:*:http://www.avitek.com:7001/myserver` specifies that any endpoint URL beginning with `http://www.avitek.com:7001/myserver` can be matched to this Service Provider, such as: `http://www.avitek.com:7001/myserver/contextA/endpointA` and `http://www.avitek.com:7001/myserver/contextB/endpointB`. If more than one Service Provider partner is discovered that is a match for the initial string, the partner with the longest initial string match is selected. This form of partner lookup string excludes the endpoint URL from being added as an Audience URI in the generated assertion.

> **Note:**
>
> Configuring one or more partner lookup strings for a Service Provider partner is required in order for that partner to be discovered at run time. If this partner cannot be discovered, no assertions for this partner can be generated.
>
> If you configure an endpoint URL without using the target lookup prefix, it will be handled as a conventional Audience URI that must be contained in assertions generated for this Service Provider partner. (This also enables backwards-compatibility with existing Audience URIs that may be configured for this partner.)

Specifying Default Partners

To support the need for a default Service Provider partner entry, one or more of the default partner's Audience URI entries may contain a wildcard match that works for all targets. The actual wildcard URI may depend on the specific format used by the web service run time. For example:

- `target:*:http://`
- `target:*:https://`

Management of Partner Certificates

The SAML 2.0 Credential Mapping provider manages a set of trusted certificates for each partner configured for web single sign-on. Whenever a signed authentication or artifact request is received during a message exchange with a partner, the trusted certificate is used to verify the partner's signature. Partner certificates are used for the following purposes:

- To validate trust when the SAML 2.0 Credential Mapping provider receives a signed authentication request or artifact request.
- To validate trust in a Service Provider partner that is retrieving a SAML artifact from the Artifact Resolution Service (ARS) via an SSL connection.

From WebLogic Remote Console, you can view a web single sign-on Service Provider partner's signing certificate and transport layer client certificate in the partner management pages of the configured SAML 2.0 Credential Mapping provider.

Java Interface for Configuring Service Provider Partner Attributes

For details about the available operations on web service partners, see the com.bea.security.saml2.providers.registry.Partner Java interface in the *Java API Reference for Oracle WebLogic Server*.

9

Configuring the Certificate Lookup and Validation Framework

Oracle WebLogic Server may receive digital certificates as part of Web services requests, two-way SSL, or other secure interactions. To validate these certificates, WebLogic Server includes a Certificate Lookup and Validation (CLV) framework.

This chapter includes the following sections:

Overview of the Certificate Lookup and Validation Framework

The key elements of the CLV framework are the CertPathBuilder and the CertPathValidators. The CLV framework requires one and only active CertPathBuilder which, given a reference to a certificate chain, finds the chain and validates it, and zero or more CertPathValidators which, given a certificate chain, validates it.
When WebLogic Server receives a certificate, the CLV framework uses the security provider configured as the CertPathBuilder to look up and validate the certificate chain. If the certificate chain is found and valid, the framework then calls each configured CertPathValidator, in the order the administrator configured them, to perform extra validation on the chain. The chain is only valid if the builder and all the validators successfully validate it.

A chain is valid only if all of the following are true:

- The certificates in the chain have signed each other properly.
- The chain terminates in a certificate that is one of the server's trusted CAs.
- The chain honors the basic constraints rules (for example, no certificate in the chain has been issued by a certificate that is not allowed to issue certificates).
- The certificates in the chain have not expired.

WebLogic Server includes two CLV security providers: the WebLogic CertPath provider (which acts as both a CertPathBuilder and a CertPathValidator), described in CertPath Provider. and the Certificate Registry, described in Certificate Registry. Use just the WebLogic CertPath provider if you want to use trusted CA-based validation of the full certificate chain. Use just the Certificate Registry if you want only to validate that certificates are registered. Use both, designating the Certificate Registry as the current builder, if you want to use both types of validation.

For more information about certificate lookup and validation, see Configuring Keystores.

CLV Security Providers Provided by WebLogic Server

WebLogic Server supports two CLV security providers: the WebLogic CertPath provider and the Certificate Registry. These providers are described in the following sections:

CertPath Provider

The default security realm in WebLogic Server is configured with the WebLogic CertPath provider. The CertPath provider serves two functions: CertPathBuilder and CertPathValidator. The CertPath provider receives an end certificate or a certificate chain. It uses the server's list of trusted CAs to complete the certificate chain, if necessary. After building the chain, the CertPath provider validates the chain, checking the signatures in the chain, ensuring that the chain has not expired, checking the chain's basic constraints, and verifying that the chain terminates in a certificate issued by one of the server's trusted CAs.

The WebLogic CertPath provider requires no configuration, other than its Current Builder attribute, which indicates whether the CertPath provider should be used as the active certificate chain builder.

Certificate Registry

The Certificate Registry is a security provider that allows you to explicitly register the list of trusted certificates that are allowed to access WebLogic Server. If you configure a Certificate Registry as part of your security realm, then only certificates that are registered in the Certificate Registry will be considered valid. The Certificate Registry provides an inexpensive mechanism for performing revocation checking. By removing a certificate from the Certificate Registry, you can invalidate a certificate immediately. The registry is stored in the embedded LDAP server.

The Certificate Registry is both a CertPath Builder and a CertPath Validator. In either case, the Certificate Registry ensures that the chain's end certificate is stored in the registry, but does no other validation. If you use the Certificate Registry as your security realm's CertPath Builder and you also want to use the WebLogic CertPath provider or another security provider to perform full chain validation, make sure that you register the intermediate and root CAs in each server's trust keystore, and the end certificates in the Certificate Registry.

The default security realm in WebLogic Server does not include a Certificate Registry. Once you configure a Certificate Registry, you can use WebLogic Remote Console to add, remove, and view certificates in the registry. You can export a certificate from a keystore to a file, using the Java keytool utility. You can import a certificate that is a PEM or DER file in the file system into the Certificate Registry using the console. You can also use WebLogic Remote Console to view the contents of a certificate, including its subject DN, issuer DN, serial number, valid dates, fingerprints, etc.

See Configure a Certification Path Provider in *Oracle WebLogic Remote Console Online Help*.

Part III
Configuring Authentication Providers

In Oracle WebLogic Server, Authentication providers are used to prove the identity of users or system processes. Authentication providers also remember, transport, and make identity information available to various components of a system, by means of subjects, when needed.

WebLogic Server includes several Authentication providers for accessing common identity stores, such as LDAP systems, DBMS systems, and more.

This part explains how to configure the Authentication providers included in WebLogic Server.

10

About Configuring the Authentication Providers in WebLogic Server

Most Authentication providers provided by Oracle WebLogic Server work in similar fashion: given a username and password credential pair, the provider attempts to find a corresponding user in the provider's data store. These Authentication providers differ primarily in what they use as a data store: one of many available LDAP servers, a SQL database, or other data store. In addition to these username/password based security providers, WebLogic Server includes identity assertion Authentication providers, which use certificates or security tokens, rather than username/password pairs, as credentials.
This chapter includes the following topics:

Choosing an Authentication Provider

The WebLogic Server security architecture supports password-based and certificate-based authentication, HTTP certificate-based authentication proxied through an external Web server, perimeter-based authentication, and authentication based on multiple security token types and protocols. WebLogic Server includes the following Authentication providers to support these authentication types:

- The *WebLogic Authentication provider*, also known as the *DefaultAuthenticator*, accesses user and group information in WebLogic Server's embedded LDAP server.
- The *Oracle Internet Directory Authentication provider* accesses users and groups in Oracle Internet Directory, an LDAP version 3 directory.
- *LDAP Authentication providers* access external LDAP stores. You can use an LDAP Authentication provider to access any LDAP server. WebLogic Server provides LDAP Authentication providers already configured for Open LDAP and Microsoft Active Directory servers.
- *RDBMS Authentication providers* access external relational databases. WebLogic Server provides three RDBMS Authentication providers: SQL Authenticator, Read-only SQL Authenticator, and Custom RDBMS Authenticator.
- The *WebLogic Identity Assertion provider* validates X.509 and IIOP-CSIv2 tokens and optionally can use a user name mapper to map that token to a user in a WebLogic Server security realm.
- The *SAML Authentication provider*, which authenticates users based on Security Assertion Markup Language 1.1 (SAML) assertions.
- The *Negotiate Identity Assertion provider*, which uses Simple and Protected Negotiate (SPNEGO) tokens to obtain Kerberos tokens, validates the Kerberos tokens, and maps Kerberos tokens to WebLogic users.
- The *SAML Identity Assertion provider*, which acts as a consumer of SAML security assertions. This enables WebLogic Server to act as a SAML destination site and supports using SAML for single sign-on.

- The *Oracle Identity Cloud Integrator* provider integrates with the Oracle Identity Cloud Service. The Oracle Identity Cloud Integrator provider combines authentication and identity assertion in a single provider. You can authenticate using username and passwords or Oracle Identity Cloud Service identity tokens.
- The *WebLogic OpenID Connect* provider adds support for using external authorization servers based on the OAuth and OpenID Connect standards. It combines authentication and identity assertion services in a single provider.

In addition, you can use:

- Custom (non-WebLogic) Authentication providers, which offer different types of authentication technologies.
- Custom (non-WebLogic) Identity Assertion providers, which support different types of tokens.

Using More Than One Authentication Provider

Each security realm must have at least one Authentication provider configured. The WebLogic Security Framework supports multiple Authentication providers (and thus multiple LoginModules) for multipart authentication. Therefore, you can use multiple Authentication providers as well as multiple types of Authentication providers in a security realm. For example, if you want to use both a retina-scan and a username/password-based form of authentication to access a system, you configure two Authentication providers.
How you configure multiple Authentication providers can affect the overall outcome of the authentication process. Configure the JAAS Control Flag for each Authentication provider to set up login dependencies between Authentication providers and allow single-sign on between providers. See Setting the JAAS Control Flag Option.

Authentication providers are called in the order in which they were configured in the security realm. Therefore, use caution when configuring Authentication providers. You can use WebLogic Remote Console to re-order the configured Authentication providers, thus changing the order in which they are called. See Changing the Order of Authentication Providers.

Setting the JAAS Control Flag Option

When you configure multiple Authentication providers, use the JAAS Control Flag for each provider to control how the Authentication providers are used in the login sequence. You can set the JAAS Control Flag in the WebLogic Remote Console. See Set the JAAS Control Flag in *Oracle WebLogic Remote Console Online Help*. You can also use the WebLogic Scripting Tool or Java Management Extensions (JMX) APIs to set the JAAS Control Flag for an Authentication provider.

JAAS Control Flag values are:

- REQUIRED—The Authentication provider is always called, and the user must pass its authentication test. However, regardless of whether authentication succeeds or fails, authentication still continues down the list of providers.
- REQUISITE—The Authentication provider is always called, and the user is required to pass its authentication test.
  - If authentication succeeds, subsequent providers are executed but can fail (except for REQUIRED Authentication providers).
  - If authentication fails, control is returned to the caller and no subsequent Authentication provider down the list is executed.

- SUFFICIENT—The user is not required to pass the authentication test of the Authentication provider.
  - If authentication succeeds, control is returned to the caller and no subsequent Authentication provider down the list is executed.
  - If authentication fails, authentication continues down the list of providers.

  Any REQUIRED or REQUISITE Authentication provider in the list must pass its own authentication test. If no REQUIRED or REQUISITE Authentication provider is in the list, then the authentication test of at least one OPTIONAL or SUFFICIENT Authentication provider must pass.
- OPTIONAL—The user is not required to pass the authentication test of the Authentication provider. Regardless of whether authentication succeeds or fails, authentication continues down the list of providers.

The overall authentication of the user succeeds only if all REQUIRED and REQUISITE Authentication providers configured in the realm succeed. Note also:

- If a SUFFICIENT Authentication provider is configured and succeeds, then only the REQUIRED and REQUISITE Authentication providers prior to that SUFFICIENT Authentication provider need to have succeeded for the overall authentication to succeed.
- If no REQUIRED or REQUISITE Authentication providers are configured in the security realm, then at least one SUFFICIENT or OPTIONAL Authentication provider must succeed.

When additional Authentication providers are added to an existing security realm, by default the Control Flag is set to OPTIONAL. If necessary, change the setting of the Control Flag and the order of Authentication providers so that each Authentication provider works properly in the authentication sequence.

> **Note:**
>
> As part of the startup process, WebLogic Server must be able to initialize all security providers that are configured in the security realm, including any Authentication providers that have a JAAS Control Flag set to OPTIONAL. If the initialization process for any security provider cannot be completed, WebLogic Server fails to boot, and an error message similar to the following is displayed:
>
> ```
> <BEA-090870> <The realm "myrealm" failed to be loaded:
> ```

Changing the Order of Authentication Providers

The order in which WebLogic Server calls multiple Authentication providers can affect the overall outcome of the authentication process. The Authentication Providers table lists the authentication providers in the order in which they will be called. By default, Authentication providers are called in the order in which they were configured. You can use WebLogic Remote Console to change the order of Authentication providers.

11

Configuring the WebLogic Authentication Provider

The WebLogic Authentication provider (also called the DefaultAuthenticator) uses Oracle WebLogic Server's embedded LDAP server to store user and group membership information and, optionally, a set of user attributes such as phone number, email address, and so on. This provider allows you to create, modify, list, and manage users and group membership in the WebLogic Remote Console. By default, most configuration options for the WebLogic Authentication provider are already defined.
This chapter includes the following sections:

About the WebLogic Authentication Provider

The WebLogic Authentication provider is configured in the default security realm with the name `DefaultAuthenticator`. You need to configure a WebLogic Authentication provider only when creating a new security realm. However, note the following:

- The WebLogic Authentication provider is configured in the default security realm with the name `DefaultAuthenticator`.
- User and group names in the WebLogic Authentication provider are case insensitive. For information about creating and managing users and groups in the WebLogic Remote Console, see Users and Groups in *Oracle WebLogic Remote Console Online Help*.
- Ensure that all user names are unique.
- Specify the minimum length of passwords defined for users that are stored in the embedded LDAP server, which you can by means of the **Minimum Password Length** option that is available on the **Configuration** > **Provider Specific** page for the WebLogic Authentication provider.
- Users in the WebLogic Authentication provider can be modified to include a set of attributes. See Setting User Attributes.
- If you are using multiple Authentication providers, set the JAAS Control Flag to determine how the WebLogic Authentication provider is used in the authentication process. See Using More Than One Authentication Provider.

Setting User Attributes

After you have defined a user in the WebLogic Authentication provider, you can set or modify one more of the attributes for that user, such as contact details, geographical location, and so on.These attributes, listed and described in Table 11-1, conform to the user schema for representing individuals in the inetOrgPerson LDAP object class, which is described in RFC 2798.

Table 11-1 Attributes that Can Be Set for a User

Attribute	Description
c	Two-letter ISO 3166 country code
departmentnumber	Code for department to which the user belongs
displayname	Preferred name of the user
employeenumber	Numeric or alphanumeric identifier assigned to the user
employeetype	Type of employment, which represents the employer to employee relationship
facsimiletelephonenumber	Facsimile (fax) telephone number
givenname	First name; that is, not surname (last name) or middle name
homephone	Home telephone number
homepostaladdress	Home postal address
l	Name of a locality, such as a city, county or other geographic region
mail	Electronic address of user (email)
mobile	Mobile telephone number
pager	Pager telephone number
postaladdress	Postal address at location of employment
postofficebox	Post office box
preferredlanguage	User's preferred written or spoken language
st	Full name of state or province
street	Physical location of user
telephonenumber	User's telephone number in organization
title	Title representing user's job function

When you set a value for an attribute, the attribute is added for the user. Likewise, if you subsequently delete the value of an attribute, the attribute is removed for the user. The set of available attributes is limited to the preceding list, however. The attribute names cannot be customized.

These attributes can be managed for a user by operations on the `UserAttributeEditorMBean`, or viewed using the operations on the `UserAttributeReaderMBean`.

12

Configuring LDAP Authentication Providers

Oracle WebLogic Server includes LDAP Authentication providers to give access to user information contained in several common LDAP identity stores.

This chapter includes the following sections:

LDAP Authentication Providers Included in WebLogic Server

WebLogic Server includes LDAP Authentication providers for identity stores such as Oracle Internet Directory, Oracle Unified Directory, and more. The full set of included LDAP Authentication providers are as follows:

- Oracle Internet Directory Authentication provider
- Oracle Unified Directory Authentication provider
- Active Directory Authentication provider
- Open LDAP Authentication provider
- Generic LDAP Authentication provider

Each LDAP Authentication provider stores user and group information in an external LDAP server. They differ primarily in how they are configured by default to match typical directory schemas for their corresponding LDAP server. For information about configuring the Oracle Internet Directory provider to match the LDAP schema for user and group attributes, see Configuring Users and Groups in the Oracle Internet Directory Authentication Provider.

WebLogic Server does not support or certify any particular LDAP servers. Any LDAP v2 or v3 compliant LDAP server should work with WebLogic Server. The following LDAP directory servers have been tested:

- Oracle Internet Directory

- Oracle Unified Directory
- Active Directory shipped as part of the Microsoft Windows platform
- Open LDAP

An LDAP Authentication provider can also be used to access other LDAP servers. However, you must either use the LDAP Authentication provider (`LDAPAuthenticator`) or choose a pre-defined LDAP provider and customize it. See Accessing Other LDAP Servers.

> **Note:**
>
> The Active Directory Authentication provider also supports Microsoft Active Directory Application Mode (ADAM) as a standalone directory server.

Requirements for Using an LDAP Authentication Provider

If an LDAP Authentication provider is the only configured Authentication provider for a security realm, you must have the `Admin` role to boot WebLogic Server and use a user or group in the LDAP directory. Do one of the following in the LDAP directory:

- By default in WebLogic Server, the `Admin` role includes the `Administrators` group. Create an `Administrators` group in the LDAP directory, if one does not already exist. Make sure the LDAP user who will boot WebLogic Server is included in the group.

  The Active Directory LDAP directory has a default group called `Administrators`. Add the user who will be booting WebLogic Server to the `Administrators` group and define Group Base Distinguished Name (DN) so that the Administrators group is found.
- If you do not want to create an `Administrators` group in the LDAP directory (for example, because the LDAP directory uses the `Administrators` group for a different purpose), create a new group (or use an existing group) in the LDAP directory and include the user from which you want to boot WebLogic Server in that group. Assign that group the `Admin` role.

> **Note:**
>
> If the LDAP user who boots WebLogic Server is not properly added to a group that is assigned to the `Admin` role, and the LDAP authentication provider is the only authentication provider with which the security realm is configured, WebLogic Server cannot be booted.

Configuring an LDAP Authentication Provider: Main Steps

After you choose an LDAP Authentication provider that matches your LDAP server, you need to enable communication between the provider and the LDAP server, configure the way in which user and group information can be accessed in the LDAP server, and configure settings that optimize the performance of the LDAP Authentication provider.
To configure an LDAP Authentication provider, complete the following main steps:

1. Choose an LDAP Authentication provider that matches your LDAP server and create an instance of the provider in your security realm. See the following topics:

- If you are using WebLogic Remote Console, see Configure an Authentication or Identity Assertion Provider in *Oracle WebLogic Remote Console Online Help*.
- If you are using the WebLogic Scripting Tool (WLST), see Managing Security Data (WLST Online) in *Understanding the WebLogic Scripting Tool*. This section also explains how to use WLST to switch from one LDAP authentication provider to another.

2. Configure the provider-specific attributes of the LDAP Authentication provider, which you can do through WebLogic Remote Console. For each LDAP Authentication provider, attributes are available to:

   a. Enable communication between the LDAP server and the LDAP Authentication provider. For a more secure deployment, Oracle recommends using the SSL protocol to protect communications between the LDAP server and WebLogic Server. Enable SSL with the `SSLEnabled` attribute.

   b. Configure options that control how the LDAP Authentication provider searches the LDAP directory.

   > **Note:**
   >
   > The value you enter for `principal` must be an LDAP administrator who has the privilege to search users and groups in the corresponding LDAP server. If the LDAP administrator does not have privileges to search the LDAP server, an LDAP exception with error code 50 is generated.

   c. Specify where in the LDAP directory structure users are located.

   d. Specify where in the LDAP directory structure groups are located.

   > **Note:**
   >
   > When specifying an LDAP search filter for users or groups using the following LDAPAuthenticatorMBean attributes, wildcards are accepted but they can have a negative performance impact on the LDAP server, particularly if you use a combination of them:
   >
   > - `AllUsersFilter`
   > - `UserFromNameFilter`
   > - `AllGroupsFilter`
   > - `GroupFromNameFilter`
   >
   > For example, the following filter expression combines five wildcarded conditions, each condition using two asterisk wildcard characters:
   >
   > ```
   > (|(cn=*wall*)(givenname=*wall*)(sn=*wall*)(cn=*wall*)(mail=*wall*))
   > ```
   >
   > The preceding example filter would likely cause an unacceptable overhead on the corresponding LDAP server.
   >
   > Additionally, group names must not contain any trailing space characters.

   e. Define how members of a group are located.

f. Set the name of the global universal identifier (GUID) attribute defined in the LDAP server.

> **Note:**
>
> If you are configuring the Oracle Internet Directory Authentication provider, see Configuring Users and Groups in the Oracle Internet Directory Authentication Provider. This section explains how to match the authentication provider attributes for users and groups to the LDAP directory structure.

g. Set timeout values for the connection to the LDAP server. You can specify two timeout values: a connection timeout, and a socket timeout.

The connection timeout, specified in the `LDAPServerMBean.ConnectTimeout` attribute for all LDAP Authentication providers, has a default value of zero. This default setting specifies no timeout limit, and can result in a slowdown in WebLogic Server execution if the LDAP servers configured for an LDAP Authentication provider are unavailable. In addition, if WebLogic Server has multiple LDAP Authentication providers configured, the failure to connect to one LDAP server may block the use of the other LDAP Authentication providers.

Oracle recommends that you set the `LDAPServerMBean.ConnectTimeout` attribute on the LDAP Authentication provider to a non-zero value; for example, 60 seconds. You can set this value using either WebLogic Remote Console or WLST. You can also set this value in the `config.xml` file by adding the following configuration parameter for the LDAP Authentication provider:

```
<wls:connect-time>60</wls:connect-time>
```

> **Note:**
>
> Oracle recommends that you do not edit the `config.xml` file directly.

The socket timeout, specified in the `-Dweblogic.security.providers.authentication.ldap.socketTimeout` JVM configuration option, sets the timeout in seconds for connecting to any one LDAP server specified in the `LDAPServerMBean.Host` attribute. The default value of the socket timeout is 0, which sets no socket timeout on the connection.

For information about the appropriate values to set for the connection timeout and socket timeout values for an LDAP Authentication provider, see Configuring Failover for LDAP Authentication Providers.

3. Configure performance options that control the cache for the LDAP server. See Improving the Performance of LDAP Authentication Providers.

> **Note:**
>
> If the LDAP Authentication provider fails to connect to the LDAP server, or throws an exception, check the configuration of the LDAP Authentication provider to make sure it matches the corresponding settings in the LDAP server.

See the following topics:

Accessing Other LDAP Servers

The LDAP Authentication providers in this release of WebLogic Server are configured to work readily with the Oracle Internet Directory, Oracle Unified Directory, Active Directory, and Open LDAP servers. You can use an LDAP Authentication provider to access other types of LDAP servers. Choose either the generic LDAP Authentication provider (`LDAPAuthenticator`) or the existing LDAP provider that most closely matches the new LDAP server and customize the existing configuration to match the directory schema and other attributes for your LDAP server. If you are using Oracle Unified Directory, see Configuring an Authentication Provider for Oracle Unified Directory.

If you are using Active Directory, see Following Referrals in the Active Directory Authentication Provider.

Enabling an LDAP Authentication Provider for SSL

To configure SSL for an LDAP Authentication provider, you must create and configure a custom trust keystore for use with the LDAP server, and specify that the SSL protocol should be used by the LDAP Authentication provider when connecting to that LDAP server.

To do this, complete the following steps:

1. Configure the LDAP Authentication provider. Make sure you turn on the **SSLEnabled** option .
2. Obtain the root certificate authority (CA) certificate for the LDAP server.
3. Create a trust keystore using the preceding certificate. For example, the following example shows using the `keytool` command to create a JKS keystore named `ldapTrustKS` with the root CA certificate `rootca.pem`:

```
keytool -importcert -keystore ./ldapTrustKS -trustcacerts -alias oidtrust -
file rootca.pem -noprompt -storetype jks
```

> **Note:**
>
> When you enter the command as shown above, `keytool` prompts you to enter a password for the keystore.

For more information about creating a trust keystore, see Configuring Keystores.

4. Copy the keystore to a location from which WebLogic Server has access.
5. In WebLogic Remote Console, open the **Edit Tree** and go to **Environment**, then **Servers**, then *myServer*. On the **Security** tab, click the **Keystores** subtab.
6. If necessary, in the **Keystores** field, click **Change** to select the **Custom Identity and Custom Trust** configuration rules.
7. If the communication with the LDAP server uses 2-way SSL, configure the custom identity keystore, keystore type, and passphrase.
8. In **Custom Trust Keystore**, enter the path and file name of the trust keystore you created.
9. In **Custom Trust Keystore Type**, enter `jks`. If you created a PKCS12 keystore (`-storetype pkcs12`), enter `pkcs12` here.
10. In **Custom Trust Keystore Passphrase**, enter the password used when creating the keystore.
11. Reboot the WebLogic Server instance for changes to take effect.

See Configuring SSL. For more information about using WebLogic Remote Console to configure keystores and enable SSL, see the following topics in *Oracle WebLogic Remote Console Online Help*:

- Identity and Trust
- Set Up TLS

Dynamic Groups and WebLogic Server

Many LDAP servers have a concept of dynamic groups or virtual groups. Many LDAP servers have a concept of dynamic groups or virtual groups. These are groups that, rather than consisting of a list of users and groups, contain some policy statements, queries, or code that define the set of users that belong to the group. Even if a group is marked dynamic, users must log out and log back in before any changes in their group memberships take effect. The term *dynamic* describes the means of defining the group and not any runtime semantics of the group within WebLogic Server.

Use of GUID and LDAP DN Data in WebLogic Principals

When a user is authenticated into WebLogic Server, an authentication provider creates a Subject with a set of user and group principals, which include the user and group names, respectively. The LDAP Authentication providers included in WebLogic Server also store the global universal identifier (GUID) and LDAP distinguished name (DN) data of users and groups as attributes of those principals. By default, WebLogic Server does not use the GUID or DN data in WebLogic principals. However, if the WebLogic domain is configured to use JAAS

authorization, the GUID and DN data can be used in principal comparison operations that occur with Java policy decisions.
When configuring an LDAP Authentication provider, make sure that the name of the GUID attribute defined in the LDAP server is specified correctly for that provider. The default GUID attribute name for each LDAP Authentication provider included in WebLogic Server is listed in Table 12-1.

Table 12-1 Name of GUID Attribute for LDAP Authentication Providers in WebLogic Server

Provider	Default GUID Attribute Name
WebLogic Authentication provider	`orclguid`[1]
Oracle Internet Directory Authentication provider	`orclguid`
Active Directory Authentication provider	`objectguid`[2]
Oracle Unified Directory Authentication provider	`entryuuid`
Open LDAP Authentication provider	`entryuuid`

1 Note that the GUID attribute name for the embedded LDAP server cannot be modified, so the WebLogic Authentication provider does not have a corresponding attribute that is configurable.
2 The Active Directory Authentication provider also supports Microsoft Active Directory Application Mode (ADAM) as a standalone directory server.

For more information about how GUID and DN data in principal objects may be used, see Configuring a Domain to Use JAAS Authorization.

Configuring Users and Groups in the Oracle Internet Directory Authentication Provider

You can modify the default values in the Oracle Internet Directory Authentication provider that specify how users and groups are located in the LDAP server.

- Configuring User and Group Name Types
- Configuring Static Groups

Configuring User and Group Name Types

By default, the Oracle Internet Directory provider is configured to search users and groups in the LDAP directory using the class attribute types identified in the following table:

Table 12-2 Class Attribute Types Used for Searches

Class	Attribute	Type
User object class	user name	`cn`
Group object class	group name	`cn`

If the user name attribute type, or group name attribute type, defined in the LDAP directory structure differs from the default settings for the Authentication provider you are using, you must change those provider settings. The following sections explain how to make those changes.

> **Note:**
>
> The Oracle Internet Directory Authentication provider cannot read the name of a user or group from the LDAP server if the name contains an invalid character. Invalid characters are:
>
> - Comma (,)
> - Plus sign (+)
> - Quotes (")
> - Backslash (\)
> - Angle brackets (< or >)
> - Semicolon (;)
>
> If the provider encounters a group or user name containing an invalid character, the name is ignored. (WebLogic Server in general does not support group names containing any of these invalid characters. See Create a Group in *Oracle WebLogic Remote Console Online Help*.)

This section includes the following topics:

Changing the User Name Attribute Type

By default, the Oracle Internet Directory Authentication provider is configured with the user name attribute set to type `cn`. If the user name attribute type in the LDAP directory structure uses a different type — for example, `uid` — you must change the following Authentication provider attributes:

- `AllUsersFilter`
- `UserFromNameFilter`
- `UserNameAttribute`

For example, if the LDAP directory structure has the user name attribute type `uid`, the preceding Authentication provider attributes must be changed as shown in Table 12-3. The required changes are shown in **bold**.

Table 12-3 Changing the User Name Attribute Type for the User Object Class

Attribute Name	Default Setting	Required New Setting
`UserNameAttribute`	`cn`	**`uid`**
`AllUsersFilter`[1]	`(&(cn=*) (objectclass=person))`	`(&(`**`uid`**`=*)(objectclass=person))`
`UserFromNameFilter`	`(&(cn=%u) (objectclass=person))`	`(&(`**`uid`**`=%u)(objectclass=person))`

[1] When specifying an LDAP search filter for users or groups, wildcards are accepted. However, using multiple asterisk wildcards, particularly for a user or group name attribute, have a negative performance impact on the LDAP server.

To configure the user name attribute type, see Configure an Authentication or Identity Assertion Provider in *Oracle WebLogic Remote Console Online Help*.

Changing the Group Name Attribute Type

By default, the Oracle Internet Directory Authentication provider is configured with the group name attribute type of `cn` for the static group object class and dynamic group object class. If the group name attribute type in the LDAP directory structure is different — for example, type `uid` is used — you must change the following Authentication provider attributes:

- `AllGroupsFilter`
- `GroupFromNameFilter`
- `StaticGroupNameAttribute` (for static groups)
- `DynamicGroupNameAttribute` (for dynamic groups)

For example, if the LDAP directory structure of the group object class uses a group name attribute of type `uid`, you must change the Authentication provider attributes as shown in Table 12-4. The required changes are shown in **bold**.

Table 12-4 Required Changes for the Group Name Attribute Type

Attribute Name	Default Setting	Required Changes
`StaticGroupNameAttribute`	`cn`	**`uid`**
`DynamicGroupNameAttribute`	`cn`	**`uid`**
`AllGroupsFilter`[1]	`(&(cn=*)(|(objectclass=groupofUniqueNames)(objectclass=orcldynamicgroup)))`	`(&(`**`uid`**`=*)(|(objectclass=groupofUniqueNames)(objectclass=orcldynamicgroup)))`
`GroupFromNameFilter`[1]	`(|(&(cn=%g)(objectclass=groupofUniqueNames))(&(cn=%g)(objectclass=orcldynamicgroup)))`	`(|(&(`**`uid`**`=%g)(objectclass=groupofUniqueNames))(&(`**`uid`**`=%g)(objectclass=orcldynamicgroup)))`

[1] When specifying an LDAP search filter for users or groups, wildcards are accepted. However, using multiple asterisk wildcards, particularly for a user or group name attribute, have a negative performance impact on the LDAP server.

To configure the group name attributes, see Configure an Authentication or Identity Assertion Provider in *Oracle WebLogic Remote Console Online Help*.

Configuring Static Groups

The Oracle Internet Directory Authentication provider is configured by default with the following settings for static groups:

- Static group object class name of `groupofuniquenames`

- Static member DN attribute of type `uniquemember`

However, the directory structure of the Oracle Internet Directory LDAP server with which you are configuring this Authentication provider may instead define the following for static groups:

- Static group object class name of `groupofnames`
- Static member DN attribute of type `member`

If the LDAP database schema contains the static group object class name of `groupofnames`, and the static member DN attribute of type `member`, you need to change the Oracle Internet Directory Authentication provider attribute settings as shown in Table 12-5. The required changes are shown in **bold**.

Table 12-5 Attribute Settings for Static Groups in the Oracle Internet Directory Authentication Provider

Attribute	Default Setting	Required Changes
`StaticGroupObjectClass`	`groupofuniquenames`	**`groupofnames`**
`StaticMemberDNAttribute`	`uniquemember`	**`member`**
`AllGroupsFilter`[1]	`(&(cn=*)(|(objectclass=groupofUniqueNames)(objectclass=orcldynamicgroup)))`	`(&(cn=*)(|(objectclass=`**`groupofnames`**`)(objectclass=orcldynamicgroup)))`
`GroupFromNameFilter`[1]	`(|(&(cn=%g)(objectclass=groupofUniqueNames))(&(cn=%g)(objectclass=orcldynamicgroup)))`	`(|(&(cn=%g)(objectclass=`**`groupofnames`**`))(&(cn=%g)(objectclass=orcldynamicgroup)))`

1 When specifying an LDAP search filter for users or groups, wildcards are accepted. However, using multiple asterisk wildcards, particularly for a user or group name attribute, have a negative performance impact on the LDAP server.

To configure static groups, see Configure an Authentication or Identity Assertion Provider in *Oracle WebLogic Remote Console Online Help*.

Example of Configuring the Oracle Internet Directory Authentication Provider

Learn how to set up a sample Oracle Internet Directory Authentication provider and use a quick method to verify the configuration.
Perform the following steps to configure this provider:

1. Create a new Oracle Internet Directory Authentication provider. Using WebLogic Remote Console, follow the process described in Configure an Authentication or Identity Assertion Provider in *Oracle WebLogic Remote Console Online Help*. Choose `OracleInternetDirectoryAuthenticator` as the type.
2. Configure the new Oracle Internet Directory Authentication provider:
   a. Set the **Control Flag** as needed (REQUIRED, REQUISITE, OPTIONAL or SUFFICIENT), as described in Setting the JAAS Control Flag Option.
   b. Navigate to the Provider Specific tab.

c. Configure the Connection settings with the Oracle Internet Directory server values you want to use. The port must be the Oracle Internet Directory LDAP port. For the purpose of this example, assume the following values:

```
Host: hostname.com
Port: 3060
Principal: cn=orcladmin
Credential: password
SSLEnabled is unchecked
```

d. Configure the Users settings as per your Oracle Internet Directory configuration.

As described in Changing the User Name Attribute Type, pay particular attention to the fields **All Users Filter** and **User From Name Filter**. They **must** reflect the value of the **User Name Attribute** field.

The default value for **User Name Attribute** is `cn` and therefore the default values for the filter fields include `(&(cn=)...)` and `(&(*cn=%u)...)`, respectively. If you change the **User Name Attribute** value, you **must** replace it accordingly in the filter fields as well.

> **Note:**
>
> If there are any leading or trailing white spaces in these filter field values, the users list may not be properly fetched from Oracle Internet Directory and you may not be able to authenticate using the Oracle Internet Directory Authentication provider.

For the purpose of example, assume the following values. Key changes are marked in **bold**.

```
User Base DN: cn=Users,dc=us,dc=oracle,dc=com
All Users Filter: (&(uid=*)(objectclass=person))
User From Name Filter: (&(uid=%u)(objectclass=person))
User Seearch Scope: subtree
User Name Attribute: uid
User Object Class: person
```

e. Configure the Groups settings as per your Oracle Internet Directory configuration.

As described in Changing the Group Name Attribute Type, by default the Oracle Internet Directory Authentication provider is configured with the group name attribute type of `cn` for the static group object class and dynamic group object class. If the group name attribute type in the LDAP directory structure is different, you must change other Authentication provider attributes to match.

In addition, as described in Configuring Static Groups, the Oracle Internet Directory Authentication provider is configured by default with a Static group object class name of `groupofuniquenames` and a Static member DN attribute of type `uniquemember`.

If the LDAP database schema instead contains the static group object class name of `groupofnames`, and the static member DN attribute of type `member`, you need to change the attribute settings as shown in Table 12-5.

For the purpose of example, assume the following values. Key values that must match are marked in **bold**.

```
Group Base DN: cn=Groups,dc=us,dc=oracle,dc=com
All Groups Filter: (&(cn=*)(objectclass=groupofUniqueNames))
```

```
Static Group Name Attribute: cn
Static Group Object Class: groupofuniquenames
Static Member DN attribute: uniquemember
Static Group DNs from Member DN Filter: (&(uniquemember=%M)
(objectclass=groupofuniquenames))

Dynamic Group Name Attribute: (empty)
Dynamic Group Object Class:   (empty)
Dynamic Member URL Attribute: (empty)
User Dynamic Group DN Attribute:  (empty)
```

f. Configure all other sections as needed, using Configuring an LDAP Authentication Provider: Main Steps for guidance. In this example, all of the default values are appropriate.

g. Save your changes.

3. If needed, order the providers to make the Oracle Internet Directory Authentication provider first in the list.
4. Restart the WebLogic Server to complete the changes.
5. Verify the setup.

Configuring Failover for LDAP Authentication Providers

You can configure an LDAP provider to work with multiple LDAP servers and enable failover if one LDAP server is not available. Use the `LDAPAuthenticatorMBean.Host` attribute to specify the names of the additional LDAP servers. Each host name may include a trailing space character and a port number. In addition, set the Parallel Connect Delay and Connection Timeout attributes for the LDAP Authentication provider

- Parallel Connect Delay—Specifies the number of seconds to delay when making concurrent attempts to connect to multiple servers. An attempt is made to connect to the first server in the list. The next entry in the list is tried only if the attempt to connect to the current host fails. This setting might cause your application to block for an unacceptably long time if a host is down. If the value is greater than 0, another connection setup thread is started after the specified number of delay seconds has passed. If the value is 0, connection attempts are serialized.
- Connection Timeout—Specifies the maximum number of seconds to wait for the connection to the LDAP server(s) to be established. If the value is set to 0, the default, there is no maximum time limit and WebLogic Server waits until the TCP/IP layer times out to return a connection failure.

  If multiple hosts are set in the Host attribute, the connection timeout controls the total timeout value for attempts to connect to *all* the specified hosts.

  Oracle recommends setting the connection timeout to a value of at least 60 seconds, depending upon the configuration of TCP/IP.
- Socket Timeout—Specifies the maximum number of seconds to wait for the connection to any *one* host specified in the Host attribute. The socket timeout is specified only using the `-Dweblogic.security.providers.authentication.ldap.socketTimeout=`*`seconds`* security parameter for the JVM in which WebLogic Server runs. The default value of the socket timeout is 0, which sets no socket timeout.

  Note that setting the socket timeout is not available in WebLogic Remote Console. For information about the options for configuring WebLogic Server security parameters, see Security in *Command Reference for Oracle WebLogic Server*.

The following examples present scenarios that occur when an LDAP Authentication provider is configured for LDAP failover:

LDAP Failover Example 1

In the following scenario, an LDAP Authentication provider is configured with three servers in its Host attribute: `directory.knowledge.com:1050`, `people.catalog.com`, and `199.254.1.2`. The status of the LDAP servers is as follows:

- `directory.knowledge.com:1050` is down
- `people.catalog.com` is up
- `199.254.1.2` is up

WebLogic Server attempts to connect to `directory.knowledge.com`. After three seconds, or the socket connection throws an exception, the connect attempt times out and WebLogic Server attempts to connect to the next specified host (`people.catalog.com`). WebLogic Server then uses `people.catalog.com` as the LDAP Server for this connection. Otherwise, after another three seconds, WebLogic Server tries to connect to `199.254.1.2`. This process continues, but will fail if the overall LDAP server connection process exceeds 10 seconds.

Table 12-6 LDAP Configuration Example 1

LDAP Option	Value
Host	`directory.knowledge.com:1050 people.catalog.com 199.254.1.2`
Parallel Connect Delay	`0`
Connect Timeout	`10`
Socket Timeout	`3`

LDAP Failover Example 2

In the following scenario, WebLogic Server attempts to connect to `directory.knowledge.com`. After 1 second (specified by the Parallel Connect Delay attribute), the connect attempt times out and WebLogic Server tries to connect to the next specified host (`people.catalog.com`) and `directory.knowledge.com` at the same time. If the connection to `people.catalog.com` succeeds, WebLogic Server uses `people.catalog.com` as the LDAP Server for this connection. WebLogic Server cancels the connection to `directory.knowledge.com` after the connection to `people.catalog.com` succeeds.

Table 12-7 LDAP Configuration Example 2

LDAP Option	Value
Host	`directory.knowledge.com:1050 people.catalog.com 199.254.1.2`
Parallel Connect Delay	`1`
Connect Timeout	`10`
Socket Timeout	`3`

Configuring an Authentication Provider for Oracle Unified Directory

Use WebLogic Remote Console to configure the Oracle Unified Directory Authentication provider. Complete the following steps:

1. Follow the process described in Configure an Authentication or Identity Assertion Provider in *Oracle WebLogic Remote Console Online Help*. Choose `OracleUnifiedDirectoryAuthenticator` as the type.
2. Configure the connection attributes for Oracle Unified Directory, as well as any other attributes as appropriate.
3. In the **GUID Attribute** field, make sure `entryuuid` is displayed.
4. Click **Save**.

> **Note:**
>
> After you configure the Oracle Unified Directory Authentication provider and subsequently log in to WebCenter as the LDAP user configured for that provider, you might receive a `WCS` error stating that the user is not found in the identity store. You receive this error if the DefaultAuthenticator provider in your security realm is set to REQUIRED. As a workaround, change the JAAS Control Flag for the DefaultAuthenticator provider to SUFFICIENT. See Setting the JAAS Control Flag Option.

Following Referrals in the Active Directory Authentication Provider

If Active Directory uses LDAP referrals, you must configure the Active Directory Authentication provider to follow those referrals by making sure that the `LDAPServerMBean.FollowReferrals` attribute is enabled. This attribute is enabled by default, but Oracle recommends that you make sure it is specifically enabled.
You can enable this attribute using WLST or WebLogic Remote Console.

Improving the Performance of LDAP Authentication Providers

WebLogic Server supports the use of several ways to improve the performance of LDAP Authentication providers, such as optimizing settings for the group membership caches, connection pool size, and user cache.

To improve the performance of LDAP Authentication providers:

- Optimize the group membership caches used by the LDAP Authentication providers. See Optimizing the Group Membership Caches.
- Optimize the connection pool size and user cache. See Optimizing the Connection Pool Size and User Cache.
- Expose the Principal Validator cache for the security realm and increase its thresholds. See Optimizing the Principal Validator Cache.

- If you are using the Active Directory Authentication provider, configure it to perform group membership lookups using the `tokenGroups` option. The `tokenGroups` option holds the entire flattened group membership for a user as an array of system ID (SID) values. The SID values are specially indexed in the Active Directory and yield extremely fast lookup response. See Configuring the Active Directory Authentication Provider to Improve Performance .
- If you are using the generic LDAP Authentication provider, you can use the `LDAPAuthenticatorMBean` API to analyze hit/miss statistics collected from the group membership and user caches. See Analyzing the Generic LDAP Authenticator Cache Statistics.
- When you are configuring a new LDAP Authentication provider or making changes to an existing one, an API is invoked to test the connection between this provider and the corresponding LDAP server during the configuration. See Testing the LDAP Connection During Configuration.

Optimizing the Group Membership Caches

To optimize the group membership caches for an LDAP Authentication provider, set the following attributes:

- Group Membership Searching—Available from the Provider Specific page, this attribute controls whether group searches are limited or unlimited in depth. This option controls how deeply to search into nested groups. For configurations that use only the first level of nested group hierarchy, this option allows improved performance during user searches by limiting the search to the first level of the group.
  - If a limited search is defined, Max Group Membership Search Level must be defined.
  - If an unlimited search is defined, Max Group Membership Search Level is ignored.
- Max Group Membership Search Level—Available from the Provider Specific page, this attribute controls the depth of a group membership search if Group Membership Searching is defined. Possible values are:
  - 0—Indicates only direct groups will be found. That is, when searching for membership in Group A, only direct members of Group A will be found. If Group B is a member of Group A, the members will not be found by this search.
  - Any positive number—Indicates the number of levels to search. For example, if this option is set to 1, a search for membership in Group A will return direct members of Group A. If Group B is a member of Group A, the members of Group B will also be found by this search. However, if Group C is a member of Group B, the members of Group C will not be found by this search.
- Enable Group Membership Lookup Hierarchy Caching— Available from the Performance page, this attribute indicates whether group membership hierarchies found during recursive membership lookup are cached. Each subtree found will be cached. The cache holds the groups to which a group is a member. This setting only applies if Group Membership is enabled. By default, it is disabled.
- Max Group Hierarchies in Cache—Available from the Performance page, this attribute specifies the maximum size of the Least Recently Used (LRU) cache that holds group membership hierarchies. A value of 1024 is recommended. This setting only applies if Enable Group Membership Lookup Hierarchy Caching is enabled.
- Group Hierarchy Cache TTL—Available from the Performance page, this attribute specifies the number of seconds cached entries stay in the cache. The default is 60 seconds. A value of 6000 is recommended.

In planning your cache settings, bear in mind the following considerations:

- Enabling a cache involves a trade-off of performance and accuracy. Using a cache means that data is retrieved faster, but runs the risk that the data may not be the latest available.
- The time-to-live (TTL) setting how long you are willing to accept potentially stale data. This depends a lot on your particular business needs. If you frequently changes group memberships for users, then a long TTL could mean that group related changes won't show up for a while, and you may want a short TTL. If group memberships almost never change after a user is added, a longer TTL may be fine.
- The cache size is related to the amount of memory you have available, as well as the cache TTL. Consider the number of entries that might be loaded in the span of the TTL, and size the cache in relation to that number. A longer TTL will tend to require a larger cache size.

Optimizing the Connection Pool Size and User Cache

When configuring any of the LDAP Authentication providers, you can improve the performance of the connection between WebLogic Server and the LDAP server by optimizing the size of the LDAP connection pool and user cache. To make these optimizations, complete the following steps:

1. Set the LDAP connection pool size to 100 by using either of the following methods:
   - Define the following system property in the `setDomainEnv` script, which is located in the `bin` directory of the WebLogic domain:

     ```
     -Dweblogic.security.providers.authentication.LDAPDelegatePoolSize=100
     ```

   - In WebLogic Remote Console, go to the provider specific page of the LDAP authentication provider you are configuring and specify `100` in the **Connection Pool Size** field.
2. Enable and enlarge the cache used with the LDAP server by completing the following steps in WebLogic Remote Console:
   a. On the provider specific page of the LDAP authentication provider you are configuring, make sure that the **Cache Enabled** option is enabled.
   b. In the **Cache Size** field, specify a value of 3200 KB.
   c. In the **Cache TTL** field, specify a time-to-live value that matches the **Group Hierarchy Cache TTL** value (see Optimizing the Group Membership Caches). A value of 6000 is recommended).
   d. Set the results timeout value for the LDAP server. On the current Provider Specific configuration page, specify a value of 1000 ms in the field labeled **Results Time Limit**.
3. Restart WebLogic Server for the changes to take effect.

Optimizing the Principal Validator Cache

To improve the performance of an LDAP Authentication provider, the settings of the cache used by the WebLogic Principal Validation provider can be increased as appropriate. The Principal Validator cache used by the WebLogic Principal Validation provider caches signed WLSAbstractPrincipals. To optimize the performance of the Principal Validator cache, set these attributes for your security realm:

- Enable WebLogic Principal Validator Cache—Indicates whether the WebLogic Principal Validation provider uses a cache. This setting only applies if Authentication providers in the

security realm use the WebLogic Principal Validation provider and WLSAbstractPrincipals. By default, it is enabled.

- Max WebLogic Principals In Cache—The maximum size of the Last Recently Used (LRU) cache used for validated WLSAbstractPrincipals. The default setting is 500. This setting only applies if Enable WebLogic Principal Validator Cache is enabled.

Configuring the Active Directory Authentication Provider to Improve Performance

To configure an Active Directory Authentication provider to use the `tokenGroups` option, set the following attributes:

- Use Token Groups for Group Membership Lookup—Indicates whether to use the Active Directory `tokenGroups` lookup algorithm instead of the standard recursive group membership lookup algorithm. By default, this option is not enabled.

> **Note:**
>
> Access to the tokenGroups option is required (meaning, the user accessing the LDAP directory must have privileges to read the `tokenGroups` option and the `tokenGroups` option must be in the schema for user objects).

- Enable SID to Group Lookup Caching—Indicates whether or not SID-to-group name lookup results are cached. This setting only applies if the Use Token Groups for Group Membership Lookup option is enabled.
- Max SID To Group Lookups In Cache—The maximum size of the Least Recently Used (LRU) cache for holding SID to group lookups. This setting applies only if both the Use Token Groups for Group Membership Lookup and Enable SID to Group Lookup Caching options are enabled.

Analyzing the Generic LDAP Authenticator Cache Statistics

If you are using the generic LDAP Authentication provider, then you can use the LDAPAuthenticatorMBean API to analyze hit/miss statistics collected from the group membership and user caches. To analyze cache statistics, you must enable cache collection and statistics of the cache. You can do this by using either WebLogic Remote Console or the WebLogic Scripting Tool (WLST).

- **Using WebLogic Remote Console** — To enable cache collection and statistics, perform the following steps:
  1. In the **Edit Tree**, go to **Security**, then **Realms**, then *myRealm*, then **Authentication Providers**.
  2. On the Provider Specific tab for your LDAP Authentication provider, turn on the **Cache Enabled** and **Cache Statistics Enabled** options.
  3. Save the changes. If automatic realm restart is enabled, you do not need to restart the domain after activating your changes.
- **Using the WebLogic Scripting Tool (WLST)** — Cache statistics can be accessed through a runtime MBean, LdapAuthenticatorRuntimeMBean, using the WebLogic

Scripting Tool (WLST). The following example demonstrates the use of WLST to retrieve cache statistics:

```
connect('','','t3://host:port')
Please enter your username :
Please enter your password :
...
serverRuntime()
cd("ServerSecurityRuntime/")
cd("$servername")
cd("RealmRuntimes/myrealm/AuthenticatorRuntimes/
OracleInternetDirectoryAuthenticator")
ls()

The cache statistics data:
----
-r--   GroupCacheHits                          47
-r--   GroupCacheQueries                       49
-r--   GroupCacheSize                          1
-r--   GroupCacheStatStartTimeStamp            2015-07-15 19:24:02.702
-r--   Name
OracleInternetDirectoryAuthenticator
-r--   ProviderName
OracleInternetDirectoryAuthenticator
-r--   Type
LdapAuthenticatorRuntime
-r--   UserCacheHits                           296
-r--   UserCacheQueries                        300
-r--   UserCacheSize                           2
-r--   UserCacheStatStartTimeStamp             2015-07-15 19:24:01.64
```

> **Note:**
>
> Cache statistics is not supported for the `DefaultAuthenticator` Authentication provider.

Testing the LDAP Connection During Configuration

Similar to the JDBC connection testing, WebLogic Server tests the connection between the Authentication provider and the LDAP server.

On the Provider Specific page, after you configure a new LDAP Authentication provider or make changes to an existing one, when you save your configuration changes, WebLogic Server tests the connection between this provider and the corresponding LDAP server. If the test succeeds, the configuration settings are saved and you may activate them. If the test fails, an error message is displayed indicating a problem. No configuration settings are saved.

13

Configuring RDBMS Authentication Providers

In Oracle WebLogic Server, an RDBMS Authentication provider is a username/password-based Authentication provider that uses a relational database, rather than an LDAP system, as an identity store for user, password, and group information.

This chapter includes the following sections:

About Configuring the RDBMS Authentication Providers

WebLogic Server includes RDBMS Authentication providers for SQL database and relational databases. These providers include the following:

- SQL Authenticator—Uses a SQL database and allows both read and write access to the database. This Authentication provider is configured by default with a typical SQL database schema, which you can configure to match your database's schema. See Configuring the SQL Authentication Provider.
- Read-only SQL Authenticator—Uses a SQL database and allows only read access to the database. For write access, you use the SQL database's own interface, not the WebLogic security provider. See Configuring the Read-Only SQL Authenticator.
- Custom RDBMS Authenticator—Requires you to write a plug-in class. This may be a better choice if you want to use a relational database for your authentication data store, but the SQL Authenticator's schema configuration is not a good match for your existing database schema. See Configuring the Custom DBMS Authenticator.

For information about adding an RDBMS Authentication provider to your security realm, see Configure an Authentication or Identity Assertion Provider in *Oracle WebLogic Remote Console Online Help*.

Common RDBMS Authentication Provider Attributes

All three RDBMS Authentication providers included with WebLogic Server have configuration options for setting the data source name, the Group Membership Searching and Max Group Membership Search Level attributes, and the group caching attributes. These configuration options are described in the following topics:

Data Source Attribute

The Data Source Name specifies the WebLogic Server data source to use to connect to the database.

Group Searching Attributes

The Group Membership Searching and Max Group Membership Search Level attributes specify whether recursive group membership searching is unlimited or limited, and if limited, how many levels of group membership can be searched. For example, if you specify that Group Membership Searching is LIMITED, and the Max Group Membership Search Level is 0, then the RDBMS Authentication providers will find only groups that the user is a direct member of. Specifying a maximum group membership search level can greatly increase authentication performance in certain scenarios, since it may reduce the number of DBMS queries executed during authentication. However, you should only limit group membership search if you can be certain that the group memberships you require are within the search level limits you specify.

> **Note:**
>
> If the RDBMS contains cyclic groups, or groups that are defined to contain themselves, the RDBMS Authentication provider may be unable to complete the authentication process. Setting the Group Membership Searching and Max Group Membership Search Level attributes can help limit recursive group name lookups. However, the use of RDBMS Authentication providers with cyclic groups is not supported and must be avoided.

Group Caching Attributes

You can improve the performance of RDBMS Authentication providers by caching the results of group hierarchy lookups. Use of this cache can reduce the frequency with which the RDBMS Authentication provider needs to access the database. You can configure the use, size, and duration of the cache.

Configuring the SQL Authentication Provider

If you are using the SQL Authentication provider, you configure how the provider and its associated database handle user passwords, and you configure the SQL statement attributes needed for accessing user information in the database. Configuring these attributes is described in the following sections:

- Password Attributes
- SQL Statement Attributes

Password Attributes

WebLogic Server uses the following attributes to govern how the SQL Authentication provider and its underlying database handle user passwords.

Plaintext Passwords Enabled

Use the Plaintext Passwords Enabled attribute to specify whether you can use plain text passwords.

Password Style Retained

Use the Password Style Retained attribute to control how a password is stored in the database when updating an existing user's password. If enabled, the default, the password style and algorithm used for the original password in the database are used for the new password. If disabled, the provider uses the settings specified for the Password Algorithm and Password Style attributes for the new password.

Password Style

Use the Password Style attribute to specify the password style to use when storing passwords for new users, and for updating the password of existing users if the Password Style Retained attribute is disabled. Valid options are `PLAINTEXT`, `HASHED`, or `SALTEDHASHED`. `SALTEDHASHED` is selected by default.

Password Algorithm

Use the Password Algorithm attribute to set the message digest algorithm used to hash passwords for storage.

> **Note:**
>
> The SQL authenticator uses the following formula for the `SALTEDHASHED` password: `{SCRYPT-BSC} + plain text salt + base64Encode(scrypt{salt + plain text password})`
>
> The formula shown uses the default value of `SCRYPT-BSC`. If you specify a value other than `SCRYPT-BSC` for the password algorithm, then the formula will change to use that value instead. Because the SQL Authenticator uses a string type to hold the hashed password value, this formula uses base64 encoding so that the bytes produced by the password algorithm can be stored as strings in the RDBMS tables.

Table 13-1 describes the password algorithms that WebLogic Server supports. Some of the algorithms offer two settings: a standard configuration appended with `-BSC` and a hardened, more computationally expensive configuration appended with `-ADV`.

Table 13-1 Supported Password Algorithms

Password Algorithm	Description
`SCRYPT-BSC` (Default)	`SCRYPT-BSC` is the default password algorithm. • 16,384 iterations • Block size of 16 • Parallel factor of `1` • 32-byte hash length • 16-byte salt length The Password Style attribute must be set to `SALTEDHASHED` when this password algorithm is selected.

Table 13-1 (Cont.) Supported Password Algorithms

Password Algorithm	Description
SCRYPT-ADV	• 32,768 iterations • Block size of 16 • Parallel factor of 2 • 32-byte hash length • 16-byte salt length The Password Style attribute must be set to SALTEDHASHED when this password algorithm is selected.
ARGON2-BSC	Specifically, Argon2ID. • 10 iterations • 47,104 KB memory size • Parallel factor of 1 • 32-byte hash length • 16-byte salt length The Password Style attribute must be set to SALTEDHASHED when this password algorithm is selected.
ARGON2-ADV	Specifically, Argon2ID. • 10 iterations • 94,208 KB memory size • Parallel factor of 2 • 32-byte hash length • 16-byte salt length The Password Style attribute must be set to SALTEDHASHED when this password algorithm is selected.
PBKDF2-BSC	• 210,000 iterations • 512-bit key length • 128-bit salt length This password algorithm is FIPS-140 compliant. The Password Style attribute must be set to SALTEDHASHED when this password algorithm is selected.
PBKDF2-ADV	• 400,000 iterations • 512-bit key length • 128-bit salt length This password algorithm is FIPS-140 compliant. The Password Style attribute must be set to SALTEDHASHED when this password algorithm is selected.

Table 13-1 (Cont.) Supported Password Algorithms

Password Algorithm	Description
Standard Algorithms	You can also specify any standard algorithm that is recognized by a Java Cryptography Extension (JCE) provider that is available at runtime. The Java Cryptography Architecture (JCA) defines the standard algorithm specifications, described at https://docs.oracle.com/en/java/javase/17/docs/specs/security/standard-names.html#algorithm-specifications. **Note:** While all standard algorithms are supported, for security purposes, Oracle recommends that you choose a password hashing algorithm with a work factor of at least 10,000 iterations. SHA-1 and MD based password algorithms are discouraged and should be updated where possible.

SQL Statement Attributes

SQL statement attributes specify the SQL statements used by the provider to access and edit the username, password, and group information in the database. With the default values in the SQL statement attributes, it is assumed that the database schema includes the following tables:

- users (username, password, [description])
- groupmembers (group name, group member)
- groups (group name, group description)

> **Note:**
>
> The tables referenced by the SQL statements must exist in the database; the provider will not create them. You can modify these attributes as needed to match the schema of your database. However, if your database schema is radically different from this default schema, you may need to use a Custom DBMS Authentication provider instead.

Configuring the Read-Only SQL Authenticator

The Read-Only SQL Authentication provider's configurable attributes include those that specify the SQL statements used by the provider to list the username, password, and group

information in the database. You can modify these attributes as needed to match the schema of your database.

Configuring the Custom DBMS Authenticator

The Custom DBMS Authentication provider, like the other RDBMS Authentication providers, uses a relational database as its data store for user, password, and group information. Use this provider if your database schema does not map well to the SQL schema expected by the SQL Authenticator. In addition to the attributes described in Common RDBMS Authentication Provider Attributes, the Custom DBMS Authentication provider's configurable attributes include those for the plug-in class.

Plug-In Class Attributes

A Custom DBMS Authentication provider requires that you write a plug-in class that implements the `weblogic.security.providers.authentication.CustomDBMSAuthenticatorPlugin` interface. The class must exist in the system classpath and must be specified in the Plug-in Class Name attribute for the Custom DBMS Authentication provider. Optionally, you can use the Plugin Properties attribute to specify values for properties defined by your plug-in class.

14

Configuring the SAML Authentication Provider

The Oracle WebLogic Server Security Assertion Markup Language (SAML) Authentication provider may be used in conjunction with the SAML 1.1 or SAML 2.0 Identity Assertion providers to allow virtual users to log in using SAML.

If virtual users are allowed, then the SAML Identity Asserter creates user/group principals, which permit the user to be logged in as a virtual user — a user that does not correspond to any locally-known user.

> **Note:**
>
> The SAML 1.1 Identity Assertion provider (SAML Identity Assertion provider V2), the SAML 1.1 Credential Mapping provider, and related configuration and services for SAML 1.1 federation services are deprecated as of WebLogic Server 14.1.2.0.0 and will be removed in a future release. Oracle recommends using the SAML Authentication provider with SAML 2.0 providers.

If the SAML Authentication provider is configured to run before other authentication providers, and has a JAAS Control Flag set to SUFFICIENT, this provider creates an authenticated subject using the user name and groups retrieved from a SAML assertion by the SAML Identity Assertion provider V2 or the SAML 2.0 Identity Assertion provider.

If the SAML Authentication provider is not configured, or if another authentication provider (e.g., the default LDAP Authentication provider) is configured before it and its JAAS Control Flag set is set to SUFFICIENT, then the user name returned by the SAML Identity Assertion provider is validated by the other authentication provider. In the case of the default LDAP Authentication provider, authentication fails if the user does not exist in the identity directory.

> **Note:**
>
> If you configure the SAML Authentication provider to allow virtual users to log in and gain access to a resource, make note of the following:
>
> 1. The resource must be configured with a security policy to control access. If the resource is unprotected, the subject created for the virtual user has no principals, which prevents access from being granted.
> 2. The protected resource must also use the default cookie `JSESSIONID`. If the resource uses a cookie name other than `JSESSIONID`, the subject's identity is not propagated to the resource.
>
> For information about configuring security policies, see *Securing Resources Using Roles and Policies for Oracle WebLogic Server*.

If you want groups from a SAML assertion, you must configure the SAML Authentication provider even if you want the LDAP Authentication provider to verify the user's existence.

Otherwise, the groups with which the user is associated is derived from the LDAP directory and not with the groups in the assertion.

The SAML Authentication provider creates a subject only for users whose identities are asserted by either the SAML Identity Assertion provider V2 or SAML 2.0 Identity Assertion provider. The SAML Authentication provider ignores all other authentication or identity assertion requests.

15

Configuring the Password Validation Provider

Oracle WebLogic Server includes a Password Validation provider, which is configured by default in each security realm. The Password Validation provider manages and enforces a set of configurable password composition rules, and is automatically invoked by a supported authentication provider whenever a password is created or updated for a user in the realm. When invoked, the Password Validation provider performs a check to determine whether the password meets the criteria established by the composition rules. The password is then accepted or rejected as appropriate.
This chapter includes the following sections:

About the Password Validation Provider

Several WebLogic Authentication providers can be used with the Password Validation provider. This includes the following:

- WebLogic Authentication provider
- SQL Authenticator provider
- LDAP Authentication provider
- Oracle Internet Directory Authentication Provider
- Active Directory Authentication provider
- Open LDAP Authentication provider

For information about configuring the Password Validation provider in the WebLogic Remote Console, see Configure the Password Validation Provider in *Oracle WebLogic Remote Console Online Help*.

Password Composition Rules for the Password Validation Provider

By default, the Password Validation provider is configured to require passwords that have a minimum length of eight characters. When used with one of the supported LDAP authentication providers listed in the preceding section, the Password Validation provider also requires that passwords meet the additional criteria listed in Table 15-1.

> **Note:**
>
> Passwords cannot contain a curly brace ("{") as the first character.

Table 15-1 Additional Password Composition Rules Required by Password Validation Provider When Used with an LDAP Authentication Provider

LDAP Authentication Provider	Additional Password Composition Requirement
• Oracle Internet Directory Authentication provider	At least one of the characters in the password must be numeric.
• WebLogic Authentication provider • LDAP Authentication provider • Active Directory Authentication provider • Open LDAP Authentication provider	At least one of the characters in the password must be non-alphabetic. For example, a numeric character, an asterisk (*), or an octothorpe (#).

The password composition rules you optionally can configure for the Password Validation provider include the following:

- User name policies — Rules that determine whether the password may consist of or contain the user's name, or the reverse of that name
- Password length policies — Rules for the minimum or maximum number of characters in a password (composition rules may specify both a minimum and maximum length)
- Character policies — Rules regarding the inclusion of the following characters in the password:
  - Numeric characters
  - Lowercase alphabetic characters
  - Uppercase alphabetic characters
  - Non-alphanumeric characters

> **Note:**
>
> Setting password composition rules is only one component of hardening the WebLogic Server environment against brute-force password attacks. To protect user accounts, you should also configure user lockout. User lockout specifies the number of incorrect passwords that may be entered within a given interval of time before the user is locked out of his or her account. See Protecting User Accounts.

Using the Password Validation Provider with the WebLogic Authentication Provider

To use the Password Validation provider in conjunction with the WebLogic Authentication provider, ensure that the minimum password length is the same for both providers. Set the

minimum password length for WebLogic Authentication provider using WebLogic Remote Console.
By default, the WebLogic Authentication provider requires a minimum password length of 8 characters, of which one is non-alphabetic. However, the minimum password length enforced by this provider can be customized. If the WebLogic Authentication provider and Password Validation provider are both configured in the security realm, and you attempt to create a password that does not meet the minimum length enforced by the WebLogic Authentication provider, an error is generated.

If the WebLogic Authentication provider rejects a password because it does not meet the minimum length requirement, the Password Validation provider is not called. To ensure that the Password Validator is always used in conjunction with the WebLogic Authentication provider, make sure that the minimum password length is the same for both providers.

You can set the minimum password length for WebLogic Authentication provider:

- If using WebLogic Remote Console, see Configure the Password Validation Provider in *Oracle WebLogic Remote Console Online Help*.
- If using WLST, see Using WLST to Create and Configure the Password Validation Provider.

Using the Password Validation Provider with an LDAP Authentication Provider

When the Password Validation provider and an LDAP Authentication provider are configured in the security realm, passwords are validated through two separate policy checks: one from Password Validation provider, and the other from the LDAP server, which has its own password policy check. For example, Oracle Internet Directory has its own password validation mechanism, which is controlled by the LDAP server administrator. These two password validation mechanisms are separate, and each has its own set of password composition rules. If the composition rules are inconsistent, failures may occur in WebLogic Remote Console when you try to create or reset a password, even if the rules for the Password Validation provider are enforced. Therefore you should make sure that the password composition rules for the Password Validation provider do not conflict with those for the LDAP server.

Using WLST to Create and Configure the Password Validation Provider

The Password Validation provider can be administered in the security realm via a WLST script that performs operations on the `SystemPasswordValidatorMBean`, described in *MBean Reference for Oracle WebLogic Server*. You may create and configure the Password Validation provider from a single WLST script, or you may have separate scripts that perform these functions separately. The following topics explain how, providing sample WLST code snippets:

- Creating an Instance of the Password Validation Provider
- Specifying the Password Composition Rules

Creating an Instance of the Password Validation Provider

The Password Validation provider is created automatically in the security realm when you create a new domain. However, you can use WLST to create one as well, as shown in Example 15-1. This code does the following:

1. Gets the current realm and Password Validation provider.
2. Determines whether an instance of the Password Validator provider (named `SystemPasswordValidator`) has been created:
   - If the provider has been created, the script displays a message confirming its presence.
   - If the provider has not been created, the script creates it in the security realm and displays a message indicating that it has been created.

Example 15-1 Creating the System Password Validator

```
edit()
startEdit()

realm = cmo.getSecurityConfiguration().getDefaultRealm()
pwdvalidator = realm.lookupPasswordValidator('SystemPasswordValidator')

if pwdvalidator:
   print 'Password Validator provider is already created'

else:
# Create SystemPasswordValidator
 syspwdValidator = realm.createPasswordValidator('SystemPasswordValidator',
 'com.bea.security.providers.authentication.passwordvalidator.SystemPasswordValidator')
 print "---  Creation of System Password Validator succeeded!"

save()
activate()
```

Specifying the Password Composition Rules

The following example shows the WLST code that sets the composition rules for the Password Validation provider. For information about the rule attributes that can be set in this script, see the description of the `SystemPasswordValidatorMBean` in the *MBean Reference for Oracle WebLogic Server*.

```
edit()
startEdit()

# Configure SystemPasswordValidator
try:
  pwdvalidator.setMinPasswordLength(8)
  pwdvalidator.setMaxPasswordLength(12)
  pwdvalidator.setMaxConsecutiveCharacters(3)
  pwdvalidator.setMaxInstancesOfAnyCharacter(4)
  pwdvalidator.setMinAlphabeticCharacters(1)
  pwdvalidator.setMinNumericCharacters(1)
  pwdvalidator.setMinLowercaseCharacters(1)
  pwdvalidator.setMinUppercaseCharacters(1)
  pwdvalidator.setMinNonAlphanumericCharacters(1)
  pwdvalidator.setMinNumericOrSpecialCharacters(1)
  pwdvalidator.setRejectEqualOrContainUsername(true)
  pwdvalidator.setRejectEqualOrContainReverseUsername(true)
  print " --- Configuration of SystemPasswordValidator complete  ---"
except Exception,e:
        print e

save()
activate()
```

16

Configuring Identity Assertion Providers

In perimeter authentication, a system outside of Oracle WebLogic Server establishes trust through tokens, as opposed to simple authentication, where WebLogic Server establishes trust through usernames and passwords. An Identity Assertion provider verifies the tokens and performs whatever actions are necessary to establish validity and trust in the token.
This chapter includes the following sections:

About the Identity Assertion Providers

Each Identity Assertion provider is designed to support one or more token formats. WebLogic Server includes the following Identity Assertion providers:

- WebLogic Identity Asserter
- LDAP X.509 Identity Asserter
- Negotiate Identity Asserter
- SAML Identity Asserter (for SAML 1.1)
- SAML 2.0 Identity Asserter
- Oracle Identity Cloud Integrator
- WebLogic OpenID Connect

> **Note:**
>
> The following providers combine authentication and identity assertion into a single provider:
>
> - The Oracle Identity Cloud Integrator provider establishes identity (the Subject) on WebLogic Server with the authenticated user, the user's groups, and the user's application roles when the identity store is the Oracle Identity Cloud Service. See Configuring the Oracle Identity Cloud Integrator Provider.
> - The WebLogic OpenID Connect provider establishes identity (the Subject) on WebLogic Server with the authenticated user and the user's groups when the identity store is a supported OpenID provider. See Configuring the WebLogic OpenID Connect Provider.

Multiple Identity Assertion providers can be configured in a security realm, but none are required. Identity Assertion providers can support more than one token type, but only one token type per Identity Assertion provider can be active at a given time. In the **Active Type** field on the Common tab in WebLogic Remote Console, define the active token type. The WebLogic Identity Assertion provider supports identity assertion with:

- X.509 certificates
- CORBA Common Secure Interoperability version 2 (CSI v2)

  If you are using CSI v2 identity assertion, define the list of client principals in the **Trusted Client Principals** field, available from the Provider Specific tab in WebLogic Remote Console.
- `weblogic-jwt-token` tokens

  This token type is used internally for propagating identity in REST invocations of other applications in the domain, and is configured by default.

If multiple Identity Assertion providers are configured in a security realm, they can all support the same token type. However, the token can be active for only one provider at a time.

With the WebLogic Identity Assertion provider, you can use a user name mapper to map the tokens authenticated by the Identity Assertion provider to a user in the security realm. For more information about configuring a user name mapper, see Configuring a WebLogic Credential Mapping Provider.

If the authentication type in a Web application is set to `CLIENT-CERT`, the Web Application container in WebLogic Server performs identity assertion on values from request headers and cookies. If the header name or cookie name matches the active token type for the configured Identity Assertion provider, the value is passed to the provider.

The Base64 Decoding Required value on the Provider Specific page determines whether the request header value or cookie value must be Base64 Decoded before sending it to the Identity Assertion provider. The setting is enabled by default for purposes of backward compatibility; however, most Identity Assertion providers will disable this option.

See Configure an Authentication or Identity Assertion Provider in *Oracle WebLogic Remote Console Online Help*.

How an LDAP X509 Identity Assertion Provider Works

The LDAP X509 Identity Assertion provider receives an X.509 certificate, looks up the LDAP object for the user associated with that certificate, ensures that the certificate in the LDAP object matches the presented certificate, and then retrieves the name of the user from the LDAP object.

The LDAP X509 Identity Assertion provider works in the following manner:

1. An application is set up to use perimeter authentication (in other words, users or system process use tokens to assert their identity).
2. As part of the SSL handshake, the application presents it certificate. The Subject DN in the certificate can be used to locate the object that represents the user in the LDAP server. The object contains the user's certificate and name.
3. The LDAP X509 Identity Assertion provider uses the certificate in the Subject DN to construct an LDAP search to find the LDAP object for the user in the LDAP server. It gets the certificate from that object, ensures it matches the certificate it holds, and retrieves the name of the user.
4. The user name is passed to the authentication providers configured in the security realm. The authentication providers ensure the user exists and locates the groups to which the user belongs.

Configuring an LDAP X509 Identity Assertion Provider: Main Steps

Typically, if you use the LDAP X509 Identity Assertion provider, you also need to configure an LDAP Authentication provider that uses an LDAP server. The authentication provider ensures the user exists and locates the groups to which the user belongs. You should ensure both providers are properly configured to communicate with the same LDAP server.
To use an LDAP X509 Identity Assertion provider:

1. Obtain certificates for users and put them in an LDAP Server. See Configuring Keystores.

   A correlation must exist between the Subject DN in the certificate and the location of the object for that user in the LDAP server. The LDAP object for the user must also include configuration information for the certificate and the username that will be used in the Subject.
2. In your security realm, configure an LDAP X509 Identity Assertion provider. See Configure an Authentication or Identity Assertion Provider in *Oracle WebLogic Remote Console Online Help*.
3. In the WebLogic Remote Console, configure the LDAP X509 Identity Assertion provider to find the LDAP object for the user in the LDAP directory given the certificate's Subject DN.
4. Configure the LDAP X509 Identity Assertion provider to search the LDAP server to locate the LDAP object for the user. This requires the following pieces of data.
   - A base LDAP DN from which to start searching. The Certificate Mapping option for the LDAP X509 Identity Assertion provider tells the identity assertion provider how to construct the base LDAP DN from the certificate's Subject DN. The LDAP object must contain an attribute that holds the certificate.

- A search filter that only returns LDAP objects that match a defined set of options. The filter narrows the LDAP search. Configure User Filter Search to construct a search filter from the certificate's Subject DN.
- Where in the LDAP directory to search for the base LDAP DN. The LDAP X509 Identity Assertion provider searches recursively (one level down). This value must match the values in the certificate's Subject DN.

5. Configure the Certificate Attribute attribute of the LDAP X509 Identity Assertion provider to specify how the LDAP object for the user holds the certificate. The LDAP object must contain an attribute that holds the certificate.
6. Configure the User Name Attribute attribute of the LDAP X509 Identity Assertion provider to specify which of the LDAP object's attributes holds the username that should appear in the Subject DN.
7. Configure the LDAP server connection for the LDAP X509 Identity Assertion provider. The LDAP server information should be the same as the information defined for the LDAP Authentication provider configured in this security realm.
8. Configure an LDAP Authentication provider for use with the LDAP X509 Identity Assertion provider. The LDAP server information should be the same the information defined for the LDAP X509 Identity Assertion provider configured in Step 7. See Configuring LDAP Authentication Providers.

Configuring a Negotiate Identity Assertion Provider

The Negotiate Identity Assertion provider enables single sign-on (SSO) with Microsoft clients. The identity assertion provider decodes Simple and Protected Negotiate (SPNEGO) tokens to obtain Kerberos tokens, validates the Kerberos tokens, and maps Kerberos tokens to WebLogic users. The Negotiate Identity Assertion provider utilizes the Java Generic Security Service (GSS) Application Programming Interface (API) to accept the GSS security context via Kerberos.
The Negotiate Identity Assertion provider is an implementation of the Security Service Provider Interface (SSPI) as defined by the WebLogic Security Framework and provides the necessary logic to authenticate a client based on the client's SPNEGO token.

For information about adding a Negotiate Identity Assertion provider to a security realm, see Configure an Authentication or Identity Assertion Provider in *Oracle WebLogic Remote Console Online Help*. For information about using the Negotiate Identity Assertion provider with Microsoft client SSO, see Configuring Single Sign-On with Microsoft Clients

Table 16-1 Negotiate Identity Asserter Attributes

Attribute	Description
Form Based Negotiation Enabled	Indicates whether the Negotiate Identity Assertion provider and servlet filter should negotiate when a Web application is configured for FORM authentication.
Active Types	The token type this Negotiate Identity Assertion provider uses for authentication. Available token types are `Authorization.Negotiate` and `WWW-Authenticate.Negotiate`. Ensure no other identity assertion provider configured in the same security realm has this attribute set to X509.

Configuring a SAML Identity Assertion Provider for SAML 1.1

The SAML Identity Assertion provider acts as a consumer of SAML 1.1 security assertions, allowing WebLogic Server to act as a destination site for using SAML 1.1 for single sign-on. The SAML Identity Assertion provider validates SAML 1.1 assertions by checking the signature and validating the certificate for trust in the certificate registry maintained by the provider. If so, identity is asserted based on the AuthenticationStatement contained in the assertion. The SAML Identity Assertion provider can also ensure that the assertion has not been previously used. The SAML Identity Assertion provider must be configured if you want to deploy a SAML Assertion Consumer Service on a server instance.

> **Note:**
>
> The SAML 1.1 Identity Assertion provider, and related configuration and services for SAML 1.1 federation services, are deprecated as of WebLogic Server 14.1.2.0.0 and will be removed in a future release. Oracle recommends using SAML 2.0 instead.

WebLogic Server includes SAML Identity Assertion provider Version 2 for SAML 1.1. It provides greatly enhanced configuration options and is recommended for new deployments. A security realm can have not more than one SAML Identity Assertion provider, and if the security realm has both SAML Identity Assertion provider and a SAML Credential Mapping provider, both must be of the same version.

For information about how to use the SAML Identity Assertion provider in a SAML single sign-on configuration, see Configuring Single Sign-On with Web Browsers and HTTP Clients Using SAML. For general information about SAML support in WebLogic Server, see Security Assertion Markup Language (SAML) in *Understanding Security for Oracle WebLogic Server*.

The following sections explain how to configure a SAML Identity Assertion provider for SAML 1.1:

- Asserting Party Registry
- Certificate Registry

Asserting Party Registry

When you configure WebLogic Server to act as a consumer of SAML security assertions, you need to register the parties whose SAML assertions will be accepted. For each SAML Asserting Party, you can specify the SAML profile used, details about the Asserting Party, and the attributes expected in assertions received from the Asserting Party. See Configuring Asserting Parties.

Certificate Registry

The SAML Identity Assertion provider maintains a registry of trusted certificates. Whenever a certificate is received, it is checked against the certificates in the registry for validity. For each Asserting Party, the following certificates from that partner are contained in this registry:

- The certificate used for validating the signature of assertions received from this Asserting Party.

- The certificate used for verifying signatures on SAML protocol elements from this Asserting Party. This certificate must be set for the Browser/POST profile.
- The TLS/SSL certificate used for verifying trust in the Asserting Party when that partner is retrieving an artifact from the Assertion Retrieval Service (ARS) via an SSL connection.

Configuring a SAML 2.0 Identity Assertion Provider for SAML 2.0

The SAML 2.0 Identity Assertion provider acts as a consumer of SAML 2.0 security assertions, allowing WebLogic Server to act as a Service Provider for Web single sign-on, and also for WebLogic Web Services Security: accepting SAML tokens for identity through the use of the appropriate WS-SecurityPolicy assertions. The SAML 2.0 Identity Assertion provider does the following:

- Validates SAML 2.0 assertions by checking the signature and validating the certificate for trust based on data configured for the partner. The SAML 2.0 Identity Assertion provider then extracts the identity information contained in the assertion, and maps it to a local subject in the security realm.
- Optionally, extracts attribute information contained in an assertion that the SAML Authentication provider, if configured in the security realm, can use to determine the local groups in which the mapped subject belongs. (See Configuring the SAML Authentication Provider.)
- Optionally, verifies that an assertion's specified lifespan and re-use settings are properly valid, rejecting the assertion if it is expired or is not available for reuse.

Configuration of the SAML 2.0 Identity Assertion provider is controlled by setting attributes on the `SAML2IdentityAsserterMBean`. You can access the `SAML2IdentityAsserterMBean` using the WebLogic Scripting Tool (WLST), or using WebLogic Remote Console and selecting `SAML2IdentityAsserter` as an identity assertion provider. For details about these attributes, see `SAML2IdentityAsserterMBean` in the *MBean Reference for Oracle WebLogic Server*.

For information about how to use the SAML 2.0 Identity Assertion provider in a SAML single sign-on configuration, see Configuring Single Sign-On with Web Browsers and HTTP Clients Using SAML. For general information about SAML support in WebLogic Server, see Security Assertion Markup Language (SAML) in *Understanding Security for Oracle WebLogic Server*. For information about using the SAML 2.0 Identity Assertion provider in Web Service Security, see Using Security Assertion Markup Language (SAML) Tokens For Identity in *Securing WebLogic Web Services for Oracle WebLogic Server*.

For information about how to configure an Identity Provider, see Identity Provider Partners.

Identity Provider Partners

When you configure WebLogic Server to act as a Service Provider, you create and configure the Identity Provider partners from whom SAML 2.0 assertions are received and validated. Configuring an Identity Provider partner consists of establishing basic information about that partner, such as the following:

- Partner name and general description
- Name mapper class to be used with this partner
- Whether to consume attribute statements included in assertions received from this partner
- Whether the identities contained in assertions received from this partner should be mapped to virtual users
- Certificates used for validating signed assertions received from this partner

The specific information you establish depends upon whether you are configuring the partner for web single sign-on or web services. Configuring a web single sign-on Identity Provider partner also involves importing that partner's metadata file and establishing additional basic information about that partner, such as the following:

- Redirect URIs, which are URLs that, when invoked by an unauthenticated user, cause the user request to be redirected to that Identity Provider partner for authentication
- Whether SAML artifact requests received from this partner must be signed
- How SAML artifacts should be delivered to this partner

For details about configuring web single sign-on Identity Provider partners, see:

- Create and Configure Web Single Sign-On Identity Provider Partners
- Create a SAML 2.0 Web Single Sign-On Identity Provider Partner in *Oracle WebLogic Remote Console Online Help*

Configuring a web service Identity Provider partner does not use a metadata file, but does consist of establishing the following information about that partner:

- Issuer URI, which is a string that uniquely identifies this Identity Provider partner, distinguishing it from other partners in your SAML federation
- Audience URIs, which specify an audience restriction to be included in assertions received from this partner

  In WebLogic Server, the Audience URI attribute is overloaded to also include the partner lookup string, which is required by the web service run time to discover the partner. See Partner Lookup Strings Required for Web Service Partners.
- Custom name mapper class that overrides the default name mapper and that is to be used specifically with this partner

For more information about configuring web service Service Provider partners, see Create a SAML 2.0 Web Service Service Provider Partner in *Oracle WebLogic Remote Console Online Help*.

The following topics explain how to configure Identity Provider partner attributes:

- Partner Lookup Strings Required for Web Service Partners
- Management of Partner Certificates
- Java Interface for Configuring Identity Provider Partner Attributes

Partner Lookup Strings Required for Web Service Partners

For web service Identity Provider partners, you also configure Audience URIs. In WebLogic Server, the Audience URI attribute is overloaded to perform two distinct functions:

- Specify an audience restriction consisting of a target URL, per the OASIS SAML 2.0 specification.
- Contain a partner lookup string, which is required at run time by WebLogic Server to discover the Identity Provider partner for which a SAML 2.0 assertion needs to be validated.

The partner lookup string specifies an endpoint URL, which is used for partner lookup and can optionally also serve as an Audience URI restriction that must be included in the assertion received from this Identity Provider partner.

> **Note:**
>
> You must configure a partner lookup string for an Identity Provider partner so that partner can be discovered at run time by the web service run time.

Lookup String Syntax

The partner lookup string has the following syntax:

```
[target:char:]<endpoint-url>
```

In this syntax, `target:char:` is a prefix that designates the partner lookup string, where char represents one of three special characters: a hyphen, plus sign, or asterisk (`-`, `+`, or `*`). This prefix determines how partner lookup is performed, as described in Table 16-2.

> **Note:**
>
> A WebLogic Server instance that is configured in the role of Service Provider always strips off the transport, host, and port portions of an endpoint URL that is passed in to the SAML 2.0 Identity Assertion provider. Therefore, the endpoint URLs you configure in any lookup string for an Identity Provider partner should contain only the portion of the URL that follows the host and port. For example, `target:*:/myserver/xxx`.
>
> When you configure a Service Provider site, this behavior enables you to configure a single Identity Provider partner that can be used to validate all assertions for the same web service, regardless of the variations in the transport protocol (i.e., HTTP vs. HTTPS), host name, IP address, and port information across all the machines in a domain that host that web service.

Table 16-2 Identity Provider Partner Lookup String Syntax

Lookup String	Description
`target:-:<endpoint-url>`	Specifies that partner lookup is conducted for an exact match of the URL, `<endpoint-url>`. For example, `target:-:/myserver/myservicecontext/my-endpoint` specifies the endpoint that can be matched to this Identity Provider partner, for which an assertion should be validated. This form of partner lookup string excludes the endpoint URL from being added as an Audience URI for this Identity Provider partner.
`target:+:<endpoint-url>`	Specifies that partner lookup is conducted for an exact match of the URL, `<endpoint-url>`. Note: Using the plus sign (+) in the lookup string results in the endpoint URL being added as an Audience URI in the assertion received from this Identity Provider partner. Because this form of lookup string is unlikely to produce a match for an Identity Provider partner, it should be avoided.

Table 16-2 (Cont.) Identity Provider Partner Lookup String Syntax

Lookup String	Description
`target:*:<endpoint-url>`	Specifies that partner lookup is conducted for an initial-string pattern match of the URL, `<endpoint-url>`. For example, `target:*:/myserver` specifies that any endpoint URL beginning with `/myserver` can be matched to this Identity Provider, such as: `/myserver/contextA/endpointA` and `/myserver/contextB/endpointB`. If more than one Identity Provider partner is discovered that is a match for the initial string, the partner with the longest initial string match is selected. This form of partner lookup string excludes the endpoint URL from being added as an Audience URI for this Identity Provider partner.

> **Note:**
>
> Configuring one or more partner lookup strings for an Identity Provider partner is required in order for that partner to be discovered at run time. If this partner cannot be discovered, no assertions for this partner can be validated.
>
> If you configure an endpoint URL without using the target lookup prefix, it will be handled as a conventional Audience URI that must be contained in assertions received from this Identity Provider partner. (This also enables backwards-compatibility with existing Audience URIs that may be configured for this partner.)

Specifying Default Partners

To support the need for a default Identity Provider partner entry, one or more of the default partner's Audience URI entries may contain a wildcard match that works for all targets. For example, `target:*:/`.

Management of Partner Certificates

The SAML 2.0 Identity Assertion provider manages the trusted certificates for configured partners. Whenever a certificate is received during an exchange of partner messages, the certificate is checked against the certificates maintained for the partner. Partner certificates are used for the following purposes:

- To validate trust when the Service Provider site receives a signed assertion or a signed SAML artifact request.
- To validate trust in an Identity Provider partner that is retrieving a SAML artifact from the Artifact Resolution Service (ARS) via an SSL connection.

The following certificates, which are obtained from each configured Identity Provider partner, are required:

- The certificate used to verify signed SAML documents received from the partner, such as assertions and artifact requests

  The certificate used to verify signed SAML documents in web single sign-on is included in the metadata file received from the Identity Provider partner. When configuring web service Identity Provider partners, you obtain this certificate from your partner and import it into this partner's configuration via the Assertion Signing Certificate tab of the partner management page in the WebLogic Server Administration Console.

- The Transport Layer Security (TLS) client certificate that is used to verify the connection made by the partner to the local site's SSL binding for retrieving SAML artifacts (used in web single sign-on only)

  When configuring a web single sign-on Identity Provider partner, you must obtain the TLS client certificate directly from the partner. It is not automatically included in the metadata file. You can import this certificate into the configuration data for this partner via the Transport Layer Client Certificate tab of the partner management page in the WebLogic Server Administration Console.

Java Interface for Configuring Identity Provider Partner Attributes

Operations on web service partners are available in the `com.bea.security.saml2.providers.registry.Partner` Java interface.

Ordering of Identity Assertion for Servlets

When an HTTP request is sent, there may be multiple matches that can be used for identity assertion. However, identity assertion providers can only consume one active token type at a time. As a result there is no way to provide a set of tokens that can be consumed with one call. Therefore, the servlet contained in WebLogic Server is forced to choose between multiple tokens to perform identity assertion. The following ordering is used:

1. An X.509 digital certificate (signifies two-way SSL to client or proxy plug-in with two-way SSL between the client and the Web server) if X.509 is one of the active token types configured for the Identity Assertion provider in the default security realm.
2. Headers with a name in the form `WL-Proxy-Client-<TOKEN>` where `<TOKEN>` is one of the active token types configured for the Identity Assertion provider in the default security realm.

   > **Note:**
   >
   > This method is deprecated and should only be used for the purpose of backward compatibility.

3. Headers with a name in the form `<TOKEN>` where `<TOKEN>` is one of the active tokens types configured for the Identity Assertion provider in the default security realm.
4. Cookies with a name in the form `<TOKEN>` where `<TOKEN>` is one of the active tokens types configured for the Identity Assertion provider in the default security realm.

For example, if an Identity Assertion provider in the default security realm is configured to have the `FOO` and `BAR` tokens as active token types (for the following example, assume the HTTP request contains nothing relevant to identity assertion except active token types), identity assertion is performed as follows:

- If a request comes in with a `FOO` header over a two-way SSL connection, X.509 is used for identity assertion.
- If a request comes in with a `FOO` header and a `WL-Proxy-Client-BAR` header, the `BAR` token is used for identity assertion.
- If a request comes in with a `FOO` header and a `BAR` cookie, the `FOO` token will be used for identity assertion.

The ordering between multiple tokens at the same level is undefined, therefore:

- If a request comes in with a `FOO` header and a `BAR` header, then either the `FOO` or `BAR` token is used for identity assertion, however, which one is used is unspecified.
- If a request comes in with a `FOO` cookie and a `BAR` cookie, then either the `FOO` or `BAR` token is used for identity assertion, however, which one is used is unspecified.

Configuring Identity Assertion Performance in the Server Cache

When you use an Identity Assertion provider, either for an X.509 certificate or some other type of token, subjects are cached within the server. A subject is a grouping of related information for a single entity (such as a person), including an identity and its security-related configuration options. Caching subjects within the server greatly enhances performance for servlets and EJB methods with `<run-as>` tags, as well as in other situations where identity assertion is used but not cached in the HTTPSession, for example, in signing and encrypting XML documents).
To optimize the cache service that the Identity Assertion provider uses, see Optimizing the Identity Assertion Cache Service.

> **Note:**
>
> Caching can violate the desired semantics.
>
> As identity assertion performance improves, the Identity Assertion provider is less responsive to changes in the configured Authentication provider. For example, a change in the user's group will not be reflected until the subject is flushed from the cache and recreated. Setting a lower value for the command-line argument makes authentication changes more responsive at a cost to performance.

Optimizing the Identity Assertion Cache Service

To improve the performance of the Identity Assertion provider, the settings of the Identity Assertion cache service can be changed as appropriate.

To optimize the performance of the Identity Assertion cache service, set these `RealmMBean` attributes for your security realm using either the WebLogic Scripting Tool (WLST) or WebLogic Remote Console (on the configuration page for your security realm):

- `IdentityAssertionCacheEnabled` — Use this attribute to specify whether to enable cache service for the Identity Assertion provider. By default, this attribute is set to true and the caching is enabled .
- `IdentityAssertionCacheTTL` — Specify the lifetime of items in the cache by setting the maximum number of seconds a subject can live in the Identity Assertion cache. This time-to-live (TTL) value can be set only if the Identity Assertion cache is enabled. This value defaults to 300 (seconds).
- `IdentityAssertionDoNotCacheContextElements` — Specify the names of the ContextElements that are not stored in the Identity Assertion cache because these elements are present in the ContextHandler of the requests. This value is used only if `IdentityAssertionCacheEnabled` is set to true. This value defaults to an empty string list.

You can override the time-to-live (TTL) value for items in the Identity Assertion cache by using the `-Dweblogic.security.identityAssertionTTL` command-line argument. Possible values for the command-line argument are:

- Less than 0 — Disables the cache.
- 0 — Caching is enabled and the identities in the cache never time out as long as the server is running. Any changes in the user database of cached entities requires a server reboot in order for the server to pick them up.
- Greater than 0 — Caching is enabled and the cache is reset at the specified number of seconds.
  To improve the performance of identity assertion, specify a higher value for this command-line argument.

> **Note:**
>
> If the time-to-live (TTL) value is set using both the `RealmMBean` attribute `IdentityAssertionCacheTTL` and the command-line argument `-Dweblogic.security.identityAssertionTTL`, then the command-line argument takes precedence over the MBean attribute.

Authenticating a User Not Defined in the Identity Store

The WebLogic Identity Assertion provider supports the ability to authenticate a user who is not defined in the security realm's identity store. Instead, the user is created as a **virtual user** and is authenticated by means of a Subject that is populated with principals derived from attributes in the X.509 certificate passed in as part of the two-way SSL connection.
The WebLogic Identity Assertion provider is not configured by default to authenticate virtual users. However, by customizing this provider's configuration, you can enable this capability in a WebLogic domain.

> **Note:**
>
> Virtual user authentication is supported only on network ports that are configured for 2-way SSL, with listening servlets using `CLIENT-CERT` authentication.
>
> Virtual user authentication is not supported in topologies where:
>
> - SSL terminates at a front-end proxy
> - Requests are forwarded to a WebLogic Server instance in which SSL has not been enabled

The following sections explain how virtual user authentication works and give the steps to configure it in a WebLogic domain:

How Virtual User Authentication Works in a WebLogic Domain

The flow of virtual user authentication follows the standard Weblogic Server security provider JAAS authentication process. When the WebLogic Identity Assertion provider is configured to allow virtual users, a user who is not defined in the security realm's identity store can be authenticated into the domain as described in the following sequence:

1. When a user issues a request on a resource hosted in the WebLogic domain, a two-way SSL connection is established between that user and WebLogic Server.

2. The WebLogic Identity Assertion provider is invoked to authenticate the user. Because virtual users are enabled for this provider, the X.509 client certificate is passed to the provider as an X.509 type token.
3. The WebLogic Identity Assertion provider invokes the configured user name mapper to:
   a. Extract data from attributes contained in the X.509 certificate.
   b. Map the required certificate attribute data to Subject principals and credentials.
   c. Return a virtual user callback handler to the Login Module for the Virtual User Authentication provider.

   If the WebLogic Identity Assertion provider is configured to allow virtual users, and a configured user name mapper allows virtual users for the given certificate, the virtual user would be considered `allowed`.
4. The Login Module uses the virtual user callback handler to build an authenticated Subject, which is composed of principals derived from the X.509 certificate attributes. Principals derived from the certificate include the user name and can also include group name, private credentials, public credentials, and other principals, depending on the user name mapper that is used.
5. The WebLogic Security Framework invokes the Virtual User Authentication provider before any other authentication providers. Because the JAAS control flag is set to SUFFICIENT, the user is authenticated into the WebLogic domain. No identity store, such as an LDAP server, is used to validate the user or to obtain additional subject components.

Configuring Two-Way SSL and Managing Certificates Securely

Prior to configuring the WebLogic Security Framework to enable virtual users to be authenticated into a WebLogic domain, Oracle strongly recommends that you optimize the SSL configuration in your domain, and leverage the certificate validation features available in WebLogic Server to ensure that client certificates are properly trusted and validated, by completing the following steps:

1. Configure two-way SSL (SSL with client authentication), in which the server presents a certificate to the client and the client presents a certificate to the server.
2. Configure SSL to limit the minimum SSL version that is enabled for SSL connections. See Specifying the SSL/TLS Protocol Version.
3. Make sure that SSL certificate validation is properly configured for your domain. See SSL Certificate Validation.
4. Configure X.509 certificate revocation (CR) checking, which checks a certificate's revocation status as part of the SSL certificate path validation process. CR checking improves the security of certificate usage by ensuring that received certificates have not been revoked by the issuing certificate authority. See X.509 Certificate Revocation Checking.

Customizing the WebLogic Identity Assertion Provider (DefaultIdentityAsserter)

The WebLogic Identity Assertion provider, also known as the DefaultIdentityAsserter, is configured by default in WebLogic domains. To enable virtual user authentication, you can customize the default instance of this provider in your WebLogic domain, or you can create a separate instance of this provider and customize it instead.

To configure the WebLogic Identity Assertion provider so that virtual user authentication is enabled, complete the following steps:

1. Configure this provider to process X.509 token types. You can set this in the `DefaultIdentityAsserterMBean.ActiveTypes` attribute.
2. Enable virtual users. You can do this by setting the `DefaultIdentityAsserterMBean.VirtualUserAllowed` attribute to `true`.
3. Enable the default user name mapper. You can do this by setting the `DefaultIdentityAsserterMBean.UseDefaultUserNameMapper` attribute to `true`.

To customize the identity assertion provider using WebLogic Remote Console, in the **Edit Tree**, go to **Security**, then **Realms**, then *myRealm*, then **Authentication Providers** and select the default identity asserter.

WebLogic Server also supports the use of a custom user name mapper that is an implementation of the `weblogic.security.providers.authentication.X509SubjectComponentMapper` interface. If you need to map other attributes from the X.509 certificate, such as group principals, private credentials, or public credentials, a custom user name mapper might be appropriate.

Configuring the Virtual User Authentication Provider

The Virtual User Authentication Provider is not available by default in a WebLogic domain. For information about how to configure this provider, see Configuring the Virtual User Authentication Provider. Note that after you add this provider to the security realm:

1. Re-order the authentication providers so that the Virtual User Authentication provider is first.
2. Set the JAAS control flag for the Virtual User Authentication provider to `SUFFICIENT`. See Set the JAAS Control Flag in *Oracle WebLogic Remote Console Online Help*.

Using WLST to Configure Virtual User Authentication

This section provides an example of configuring virtual user authentication in a WebLogic domain. Example 16-1 shows the following:

1. Connecting to the WebLogic Server instance.
2. Creating an instance of a Virtual User Authentication provider.
3. Ordering the Virtual User Authentication provider first among the authentication providers in the security realm.
4. Enabling virtual users in the WebLogic Identity Assertion provider (DefaultIdentityAsserter).
5. Enabling the default user name mapper provided by WebLogic Server.
6. Saving and activating changes to the security realm.

Example 16-1 Configuring the Virtual User Authentication Provider and Enabling Virtual Users

```
connect('','','t3://host:port')
Please enter your username :
Please enter your password :
...
edit()
startEdit()
print 1
```

```
    cd('/SecurityConfiguration/'+domainName+'/Realms/myrealm')
    print 2
    auth=cmo.lookupAuthenticationProvider('VirtualUserAtnProvider')
    print 3
    if auth == None:
      print 4
      auth =
cmo.createAuthenticationProvider('VirtualUserAtnProvider','weblogic.security.providers.au
thentication.VirtualUserAuthenticator')
    print auth

set('AuthenticationProviders',jarray.array([ObjectName('Security:Name=myrealmVirtualUserA
tnProvider'),ObjectName('Security:Name=myrealmDefaultAuthenticator'),ObjectName('Security
:Name=myrealmDefaultIdentityAsserter')],ObjectName))
    print 5

    cd('AuthenticationProviders/DefaultIdentityAsserter')
    set('VirtualUserAllowed','true')
    print( "VirtualUserAllowed set to true" )
    set('UseDefaultUserNameMapper','true')
    print( "UseDefaultUserNameMapper set to true" )
    save()
    activate()
```

Configuring a User Name Mapper

WebLogic Server verifies the digital certificate of the Web browser or Java client when establishing a two-way SSL connection. However, the digital certificate does not identify the Web browser or Java client as a user in the WebLogic Server security realm. If the Web browser or Java client requests a WebLogic Server resource protected by a security policy, WebLogic Server requires the Web browser or Java client to have an identity. The WebLogic Identity Assertion provider allows you to enable a user name mapper that can map either of the following:

- The digital certificate of a Web browser or Java client to a user in a WebLogic Server security realm.
- Attributes contained in the X.509 certificate to Subject principals and credentials for a user that is *not* defined in the identity store of the security realm (see Authenticating a User Not Defined in the Identity Store).

The user name mapper must be an implementation of the `weblogic.security.providers.authentication.UserNameMapper` interface. This interface maps a token to a WebLogic Server user name according to whatever scheme is appropriate for your needs. By default, WebLogic Server provides a default implementation of the `weblogic.security.providers.authentication.UserNameMapper` interface. You can also write your own implementation, as described in Configuring a Custom User Name Mapper.

The WebLogic Identity Assertion provider calls the user name mapper for the following types of identity assertion token types:

- X.509 digital certificates passed via the SSL handshake
- X.509 digital certificates passed via CSIv2
- X.501 distinguished names passed via CSIv2

The default user name mapper uses the subject DN of the digital certificate or the distinguished name to map to the appropriate user in the WebLogic Server security realm. For example, the user name mapper can be configured to map a user from the Email attribute of the subject DN (`smith@example.com`) to a user in the WebLogic Server security realm (`smith`).

Use Default User Name Mapper Attribute Type and Default Username Mapper Attribute Delimiter attributes of the WebLogic Identity Assertion provider to define this information:

- Default User Name Mapper Attribute Type—The subject distinguished name (DN) in a digital certificate used to calculate a username. Valid values are: `C`, `CN`, `E`, `L`, `O`, `OU`, `S` and `STREET`. The default attribute type is `E`.
- Default User Name Mapper Attribute Delimiter—Ends the username. The user name mapper uses everything to the left of the value to calculate a username. The default delimiter is `@`.

  For example, when extracting a user name from an email address, the user name mapper uses all characters in the email address up to the @ character. Therefore, if you want the user name mapper to map a different attribute in the Subject DN — for example, the Common Name, or CN — it might be appropriate to specify a different delimiter.

To configure a user name mapper using WebLogic Remote Console, in the **Edit Tree**, go to **Security**, then **Realms**, then *myRealm*, then **Authentication Providers** and select the default identity asserter. On the provider specific tab, modify the relevant attributes.

Configuring a Custom User Name Mapper

You can also write a custom user name mapper to map a token to a WebLogic user name, or to a virtual user, according to whatever scheme is appropriate for your needs. Note the following:

- A custom user name mapper that maps a token to a WebLogic user must be an implementation of the `weblogic.security.providers.authentication.UserNameMapper` interface.
- A custom user name mapper that maps an X.509 token to Subject principals that are used to authenticate a virtual user — that is, a user that is *not* defined in the security realm identity store — must be an implementation of the `weblogic.security.providers.authentication.X509SubjectComponentMapper` interface.

  If you need to map other attributes from the X.509 certificate, such as group principals, private credentials, or public credentials, a custom user name mapper might be appropriate.

To configure a custom user name mapper using WebLogic Remote Console, in the **Edit Tree**, go to **Security**, then **Realms**, then *myRealm*, then **Authentication Providers** and select the default identity asserter. On the provider specific tab, modify the relevant attributes.

17

Configuring the Virtual User Authentication Provider

Use the Virtual User Authentication provider to authenticate users who are not defined in the identity store that is configured in the Oracle WebLogic Server security realm.

This chapter includes the following sections:

About the Virtual User Authentication Provider

You use the Virtual User Authentication provider as part of the overall capability supported in WebLogic Server to authenticate users who are not defined in the identity store with which the security realm is configured. Instead, you create a virtual user whose identity is based on select attributes contained in an X.509 certificate, such as in the Subject DN.
For complete details about configuring and using virtual user authentication in a WebLogic domain, see Authenticating a User Not Defined in the Identity Store.

> **Note:**
>
> Virtual user authentication is supported only on network ports that are configured for 2-way SSL, with listening servlets using `CLIENT-CERT` authentication.
>
> Virtual user authentication is not supported in topologies where:
>
> - SSL terminates at a front-end proxy
> - Requests are forwarded to a WebLogic Server instance in which SSL has not been enabled

Adding the Virtual User Authentication Provider to the Security Realm

You can use WebLogic Remote Console to add the Virtual User Authentication provider to a security realm.

1. To add and configure the Virtual User Authentication provider, follow the steps described in Configure an Authentication or Identity Assertion Provider in *Oracle WebLogic Remote Console Online Help*, making sure to select **VirtualUserAuthenticator** as the authentication provider type.
2. Re-order the authentication providers so that the Virtual User Authentication provider is listed first.

3. Set the JAAS control flag to `SUFFICIENT`. See Set the JAAS Control Flag in *Oracle WebLogic Remote Console Online Help*.
4. Save your changes.
5. Restart WebLogic Server.

18

Configuring the Oracle Identity Cloud Integrator Provider

The Oracle Identity Cloud Integrator provider is an authentication and identity assertion provider that accesses users, groups, and Oracle Identity Cloud Service scopes and application roles stored in the Oracle Identity Cloud Service. You can configure the provider using the WebLogic Remote Console, WLST online and WLST offline. The Oracle Identity Cloud Integrator provider supports basic authentication with the Oracle Identity Cloud Service using usernames and passwords, and perimeter authentication (identity assertion) using Oracle Identity Cloud Service tokens.
Topics in this chapter include:

About the Oracle Identity Cloud Integrator Provider

The Oracle Identity Cloud Integrator provider combines authentication and identity assertion into a single provider. The provider establishes identity (the Subject) on WebLogic Server with the authenticated user, the user's groups, and the user's application roles when the identity store is the Oracle Identity Cloud Service.

The Oracle Identity Cloud Service provides identity management, single sign-on, and identity governance for applications on-premise, in the cloud, or on mobile devices. It leverages OAuth 2.0 for authorization of custom applications and OpenID Connect to externalize authentication using federated single-sign-on. For details about the Oracle Identity Cloud Service, see http://docs.oracle.com/en/cloud/paas/identity-cloud/index.html.

You can use the Oracle Identity Cloud Integrator provider with the Oracle Identity Cloud Service as described in the following sections.

Basic Authentication

With basic authentication, the server requests a user name and password from the client and verifies that the user name and password are valid by comparing them against the authorized users in the Oracle Identity Cloud Service. Using basic authentication, users in an Oracle Identity Cloud Service tenant can log into WebLogic Remote Console, use the WebLogic Scripting Tool (WLST), or log into an application running on WebLogic Server.

Perimeter Authentication

The concept of perimeter authentication is the process of authenticating the identity of a remote user outside of the application server domain. Perimeter authentication is typically accomplished by the remote user specifying an asserted identity and some form of corresponding proof material, which extends the single sign-on concept all the way to the perimeter. The Oracle Identity Cloud Integrator provider supports perimeter authentication from users authenticated in the Oracle Identity Cloud Service using the following perimeter authentication mechanisms:

- OpenID Connect Identity (ID) tokens created by the Oracle Identity Cloud Service. The identity asserter handles the `idcs_user_assertion` HTTP header for Oracle Identity Cloud Service identity tokens by default. The user assertion (ID token) represents a user authenticated by the Oracle Identity Cloud Service and is used to map to a WebLogic Server subject containing principals with the user, group, and Oracle Identity Cloud Service application roles information.

  This functionality includes a new `IDCSAppRole` principal. See `weblogic.security.principal` in *Java API Reference for Oracle WebLogic Server*.

  Also, a new `weblogic.entitlement.rules.IDCSAppRoleName ()` predicate was added that can be used in role mapping and authorization policies.

- The `REMOTE_USER` HTTP header for Oracle Identity Cloud Service protected resources. The `REMOTE_USER` header handles users that were validated by Oracle Identity Cloud Service policy protection using the HTTP Basic header and then sent to the server using the `REMOTE_USER` HTTP header.

  In addition to setting the `REMOTE_USER` using an HTTP header, the Oracle Identity Cloud Service also specifies the user tenancy using an HTTP header.

  When Oracle Identity Cloud Service indicates that the anonymous user is accessing the service, WebLogic Server denies access to protected Java EE resources.

- Oracle Identity Cloud Service access tokens for protected resources. The access token is a credential that allows an OAuth client to access a protected resource and is used to map to a WebLogic Server Subject containing principals using the user, group, Identity Cloud Service application roles, scopes, and audience information based on the token. The provider supports access tokens using the Authorization token type and retrieves the access token from the Authorization HTTP header.

  This functionality includes two new principals, `IDCSScope` and `IDCSClient`, to support storing client and scope information in the subject. The Oracle Identity Cloud Service audience (`IDCSAudience`) is optionally stored in the public credentials of the subject. See `weblogic.security.principal` and `Class IDCSAudience` in *Java API Reference for Oracle WebLogic Server*.

  A new `weblogic.entitlement.rules.Scope ()` predicate was added that can be used in role mapping and authorization policies.

The `REMOTE_USER` and Authorization HTTP headers are not enabled by default. The `REMOTE_USER` header is not enabled by default because this header should only be sent by a trusted client. You cannot have any publicly accessible endpoints if `REMOTE_USER` is enabled on the Oracle Identity Cloud Integrator provider. When exposing both public and protected endpoints, then use of `REMOTE_USER` may leave applications and WebLogic Server open to security vulnerabilities. The Authorization HTTP header is not enabled by default because the services must be protected by the Oracle Identity Cloud Service to safely accept user information from access tokens.

If required, you must enable the Authorization and `REMOTE_USER` HTTP headers in WebLogic Remote Console using the **Active Types** attribute on the Common page for the provider or using WLST. See Enabling Authorization and REMOTE_USER Header Support: Main Steps.

To control access token processing, you can set additional configuration attributes such as `AudienceEnabled` and `ClientAsUserPrincipalEnabled`, and access token claim attributes on the Provider Specific page, or directly on the OracleIdentityCloudIntegratorMBean

Programmatic Identity Assertion

The Oracle Identity Cloud Integrator provider can be used for programmatic assertion from an OpenID Connect ID token obtained from the Oracle Identity Cloud Service. In this scenario, the application logic implements the OAuth protocol (for example, the authorization code grant flow) to obtain an ID token from Oracle Identity Cloud Service. After obtaining the ID token, the application logic uses the WebLogic Server Authentication API to assert the Oracle Identity Cloud Service user identity from the ID token. See the following specifications and API reference documents:

- The OAuth 2.0 Authorization Framework
- OpenID Connect Core 1.0 incorporating errata set 1
- `weblogic.security.services.Authentication` in *Java API Reference for Oracle WebLogic Server*

In addition, the provider can be used to validate an Oracle Identity Cloud Service user when using Oracle Platform Security Services (OPSS), Oracle Web Services Manager (OWSM), or SSO mechanisms such as SAML2.0.

Multiple Identity Store Environment

You can use the Oracle Identity Cloud Integrator provider to access the Oracle Identity Cloud Service as a single source of users, or in a hybrid environment in combination with other identity stores. As part of the WebLogic Security Framework, the Oracle Identity Cloud Integrator provider can be configured with other authentication providers in the security realm so that each provider can authenticate users in its respective identity store. For example, you can configure the Default Authenticator provider to authenticate users in the embedded LDAP server, and the Oracle Identity Cloud Integrator to authenticate users in the Oracle Identity Cloud Service. When you configure multiple Authentication providers, use the JAAS Control Flag for each provider to control how the Authentication providers are used in the login sequence. See Using More Than One Authentication Provider.

If the Oracle Identity Cloud Integrator provider is the only authentication provider configured in the security realm, an Oracle Identity Cloud Service user can boot WebLogic Server. To do so, the Oracle Identity Cloud Service user must be added to a group or granted a role that is assigned to the WebLogic Server `Admin` role. Otherwise, WebLogic Server cannot be booted. If the Oracle Identity Cloud Integrator provider fails to connect to the Oracle Identity Cloud Service, or throws an exception, make sure the configuration settings are set correctly for this provider.

Additionally, if you are setting up a single sign-on configuration, for example using SAML 2.0, you can configure the Oracle Identity Cloud Integrator provider as the authentication provider to validate the user. See Configuring Single Sign-On with Web Browsers and HTTP Clients Using SAML.

Oracle Identity Cloud Service Single Sign-On (SSO) and Logout Synchronization

When you have established your identity using the basic, perimeter, or programmatic authentication mechanisms, the provider includes an SSO synchronization filter to synchronize the Oracle Identity Cloud Service SSO session with the local container session. The SSO

synchronization filter is an implementation of the Servlet Filter. The filter intercepts each request to the container and determines whether to act on the request based on certain HTTP headers that are part of the request. The job of the filter is to ensure that the user identity in the container (tenant and user name) matches the identity of the SSO session. If the identities match, the session continues. If there is a mismatch in identities (for example a user has logged out or a session timed out), the filter invalidates the container's user session and performs a redirect to continue.

The synchronization filter is enabled by default. You can adjust the settings, if desired, in the Synchronization Filter section on the Provider Specific page in WebLogic Remote Console or by setting them on the MBeans.

Prerequisites for Configuring the Oracle Identity Cloud Integrator Provider

For WebLogic Server to authenticate users with the Oracle Identity Cloud Service, the Oracle Identity Cloud Integrator provider must be associated with an OAuth client that is registered with the Oracle Identity Cloud Service. The OAuth client allows the provider access to the Oracle Identity Cloud Service.

Before you can configure the provider you must obtain the required OAuth client information from the Oracle Identity Cloud Service. To do so, you create a trusted application in the Oracle Identity Cloud Service Console. A trusted application in the Oracle Identity Cloud Service is a type of custom application that can be accessed by multiple users and hosted in a secure and protected place (server) where the trusted application uses OAuth 2.0. Because you know where the application is hosted, you can treat that application as trusted. Creating the application in Oracle Identity Cloud Service results in the provisioning of an OAuth client.

Creating the OAuth Client: Main Steps

To create the OAuth client in the Identity Cloud Service Console:

1. Log into the Identity Cloud Service console as the Tenant Administrator.
2. Create a trusted (confidential) application. See Add a Confidential Application in *Administering Oracle Identity Cloud Service*.

   Note that the OAuth client can be used only within the specific tenant in which it was provisioned.

   In the application wizard:

   a. Enter a client name and, optionally, a description.

   b. Select **Configure this application as a client now** to configure authorization settings:

      - Select only Client Credentials as the allowed grant type. This setting is used when the authorization scope is limited to the protected resources under the control of the client or to the protected resources registered with the authorization server. The client presents its own credentials to obtain an access token.
      - Assign the client to the Identity Domain Administrator application role. To do so, select **Grant the client access to Identity Cloud Service Admin APIs** and then, in the box that is displayed, select **Identity Domain Administrator**. This enables your application to access all of the REST API endpoints and the allowed operations for those endpoints that the Identity Domain Administrator application role can access.

        Alternatively, select **Authenticator Client** to assign this role instead. Note, however, that the Authenticator Client application role supports fewer REST API

endpoints than the Identity Domain Administrator role. For a complete list of the endpoints and allowed operations that each application role can access, see Apps/App Roles endpoint in *REST API for Oracle Identity Cloud Service*.

> **Note:**
>
> Although the Identity Domain Administrator role has write access to the Oracle Identity Cloud Service user store, the WebLogic Server Oracle Identity Cloud Integrator provider does not support any update operations. Therefore, you must use the Identity Cloud Service Administration Console to modify the content of the user store.
> The Authenticator Client application role will work with the Oracle Identity Cloud Integrator Provider, but because there are fewer available Oracle Identity Cloud Service endpoints, this role may not be sufficient for other Fusion Middleware components.

3. Step through the remaining pages in the wizard and click **Finish**. Record the Client ID and Client Secret that are displayed when you create the application. You need these values when you configure the Oracle Identity Cloud Integrator provider. The attributes that you must provide when configuring the provider are described in Required Configuration Attributes.
4. Activate the application.

Required Configuration Attributes

To configure the Oracle Identity Cloud Integrator provider in WebLogic Server, you must provide the following attributes from the OAuth client:

- Tenant — Name of the primary tenant in the Oracle Identity Cloud Service where you provisioned the OAuth client.
- ClientId — The OAuth client ID used to access the Oracle Identity Cloud Service identity store.
- ClientSecret — The OAuth Client Secret (password) used to generate access tokens.
- Client tenant (Optional) — Name of the OAuth client tenant in which the Client Id resides. This attribute is not required if the Client tenant is the same as the primary tenant.

Configuring the Oracle Identity Cloud Integrator Provider: Main Steps and Examples

To configure the Oracle Identity Cloud Integrator provider, you must add the provider to the security realm and specify the configuration attributes required to enable communication between the provider and the Oracle Identity Cloud Service. You can configure the provider using WebLogic Remote Console, WLST online, or WLST offline.
The attributes required to configure the connection are described in Prerequisites for Configuring the Oracle Identity Cloud Integrator Provider. You also need to provide the Oracle Identity Cloud Service Host and Port where the Oracle Identity Cloud Service is accessible. This value for the host is the base name for the Identity Cloud Service Tenant URLs (for example `identity.example.com`) and does not include the tenant name. If TLS/SSL is enabled, be sure to use the secure port.

The main steps for configuring the provider using WebLogic Remote Console are as follows:

1. In the **Edit Tree**, go to **Security**, then **Realms**, then *myRealm*, then **Authentication Providers**.
2. Click **New**.
3. In the **Name** field, enter a name for the new provider.
4. From the **Type** drop-down list, select **Oracle Identity Cloud Integrator** as the authentication provider.
5. Click **Create**.
6. On the **Configuration** page for the new authentication provider, set the required values on the **Common** and **Oracle Identity Cloud Integrator Parameters** tabs to enable connection to the Oracle Identity Cloud Service.
7. If TLS/SSL is required, turn on the **SSL Enabled** option on the **Oracle Identity Cloud Integrator Parameters** tab.
8. Click **Save**.

Configuring the Oracle Identity Cloud Integrator Provider: WLST Online Example

You can configure the Oracle Identity Cloud Integrator Provider using WLST online in script mode by creating and executing a script that adds the provider to the security realm and configures the connection to the Oracle Identity Cloud Service.

To do so, create a WLST script, similar to the sample IdentityCloudIntegrator.py script shown in Example 18-1.

In the script, replace the required variables `idcsHost`, `idcsPort`, `clientTenant`, `clientID`, and `clientSecret` with the appropriate values for your environment. In the `connect` command in the script, replace the `username`, `password`, and `host:port` with the values for the server in the domain to which you are adding the provider. Execute the script as described in Using the WebLogic Scripting Tool in *Understanding the WebLogic Scripting Tool*

This script starts WLST online, adds the provider to the security realm, sets the provider configuration, sets the JAAS control Flag, and activates the changes.

You need to restart the server after updating the domain.

Example 18-1 Sample IdentityCloudIntegrator.py WLST Script

> **Note:**
>
> For clarity, this WLST example script shows the username and password in clear text. However, you should avoid entering clear-text passwords in WLST commands in general, and you should especially avoid saving on disk WLST scripts that include clear-text passwords. In these instances you should use a mechanism for passing encrypted passwords instead. See Security for WLST in *Understanding the WebLogic Scripting Tool*.

```
#
# Use appropriate Oracle Identity Cloud Service host, port, client tenant,
client Id, and client secret.
#
idcsHost="identity.example.com"
idcsPort=443
clientTenant="myTenant"
```

```
clientId="123456789"
clientSecret="987654321"
#
# Start WLST edit session
#
connect("","","t3://host:port")
edit()
startEdit()
#
# Add the Oracle Identity Cloud Integrator provider to the security realm
configuration where the Default
# Authenticator is the only existing authentication provider.
#
realm = cmo.getSecurityConfiguration().getDefaultRealm()
atn = realm.lookupAuthenticationProvider('IdentityCloudServiceIntegrator')
if atn == None:
  atn =
realm.createAuthenticationProvider('IdentityCloudServiceIntegrator','weblogic.
security.providers.authentication.OracleIdentityCloudIntegrator')
#
# Setup the Oracle Identity Cloud Integrator provider configuration
#
atn.setHost(idcsHost)
# Example host requires SSL. Comment out next line if using an Oracle
Identity Cloud Service instance that does not require SSL.
atn.setSSLEnabled(true)
atn.setPort(idcsPort)
atn.setTenant(clientTenant)
# If the Client Tenant == Tenant then no need to set the Client Tenant value
atn.setClientTenant(clientTenant)
atn.setClientId(clientId)
atn.setClientSecret(clientSecret)
atn.setControlFlag('SUFFICIENT')
#
# Adjust the JAAS control flag for the DefaultAuthenticator such that users
from the embedded LDAP server or the
# Oracle Identity Cloud Service are allowed to log into WebLogic Server.
#
atnDefault = realm.lookupAuthenticationProvider('DefaultAuthenticator')
if atnDefault != None:
  atnDefault.setControlFlag('SUFFICIENT')
#
# Activate changes
#
activate()
exit()
#
# Restart WebLogic Server
#
```

Configuring the Oracle Identity Cloud Integrator Provider: WLST Offline Example

You can configure the Oracle Identity Cloud Integrator provider using WLST offline by executing a series of commands that add the provider to the security realm and configure the connection to the Oracle Identity Cloud Service.

For details about using WLST offline, see Using WLST Offline to Update an Existing WebLogic Domain in *Understanding the WebLogic Scripting Tool*.

Executing these commands edits the domain as follows:

- Opens the domain configuration for editing. Be sure to replace the variables *domainDirName* and *DOMAIN_NAME* with the appropriate values for your environment. For example, if you accepted the default values when you created the domain, *domainDirName* is *ORACLE_HOME*/user_projects/domains/base_domain and *DOMAIN_NAME* is base_domain. Also be sure to provide the actual name of the security realm. In this example we used *myrealm* .
- Adds the Oracle Identity Cloud Integrator provider to the security realm configuration where the Default Authenticator is the only existing authentication provider
- Configures the connection to the Oracle Identity Cloud Service using the values you provide for idcsHost, idcsPort, clientTenant, clientID, and clientSecret.
- Adjusts the JAAS control flag for the DefaultAuthenticator such that users from the embedded LDAP server or the Oracle Identity Cloud Service are allowed to log into WebLogic Server.
- Updates and closes the domain, and exits WLST offline.

Example 18-2 WLST Offline Commands to Configure Oracle Identity Cloud Integrator Provider

```
readDomain('domainDirName')
cd('SecurityConfiguration/DOMAIN_NAME/Realm/myrealm')
create('IdentityCloudServiceIntegrator','weblogic.security.providers.authentic
ation.OracleIdentityCloudIntegrator','AuthenticationProvider')
cd('AuthenticationProviders/IdentityCloudServiceIntegrator')
# Execute the set commands needed to configure the Oracle Identity Cloud
Integrator provider host, port, tenant,
# client tenant, client id, client secret and JAAS control flag.

idcsHost="identity.example.com"
idcsPort=443
clientTenant="myTenant"
clientId="123456789"
clientSecret="987654321"

# Set attributes
set("Host",idcsHost)
set("SSLEnabled", true)
set("Port", idcsPort)
set("Tenant", clientTenant)
set("ClientTenant", clientTenant)
set("ClientId", clientId)
set("ClientSecretEncrypted", clientSecret)
set("ControlFlag", "SUFFICIENT")
# Set any other authenticators to SUFFICIENT. In this example, set the JAAS
control flag for the DefaultAuthenticator
# such that users from the embedded LDAP server or the Oracle Identity Cloud
Service are allowed to log into WebLogic Server.
cd("..")
cd("DefaultAuthenticator")
set("ControlFlag", "SUFFICIENT")
updateDomain()
```

```
closeDomain()
exit()
```

Configuring TLS/SSL for the Oracle Identity Cloud Integrator Provider

The Oracle Identity Cloud Integrator provider supports one-way SSL. To secure the connection using TLS/SSL, you need to establish trust between WebLogic Server and the Oracle Identity Cloud Service. To do so, you may need to obtain the Oracle Identity Cloud Service SSL certificate and import it into the WebLogic Server trust store.

> **Note:**
>
> If the Oracle Identity Cloud Service uses a well-known certificate authority (CA) such as Symantec, and your WebLogic domain is using Java Standard Trust, WebLogic Server trusts it by default and importing the certificate is not required. If, however, your domain is configured for custom trust, you may need to import the certificate into your trust store, regardless of whether the Oracle Identity Cloud Service is using a well-known CA.

To configure TLS/SSL:

1. On the Oracle Identity Cloud Integrator provider, set the following attributes:
   - SSLEnabled — true
   - idcsPort — the appropriate SSL port for the Oracle Identity Cloud Service , for example 443.
2. Optionally, obtain the root CA certificate from the Oracle Identity Cloud Service's server and import it into the appropriate trust store in your WebLogic Server domain.

   This step is not required if the Oracle Identity Cloud Service uses a well-known CA.
   - If your domain uses a KSS trust store, import the certificate as described in Configuring the OPSS Keystore Service for Custom Identity and Trust: Main Steps.
   - If your domain uses a JKS or PKCS12 trust store, see Importing Certificates into the Trust and Identity Keystores.
3. Configure host name verification in WebLogic Server using the wildcard host name verifier to allow WebLogic Server to accept certificates containing wildcards. The wildcard host name verifier is the default host name verifier as of WebLogic Server 14c (14.1.1.0.0). See Using the Wildcard Host Name Verifier. You can set this property in any of the following ways:
   - Configuring the property in the WebLogic Remote Console as described in Enable Host Name Verification in *Oracle WebLogic Remote Console Online Help*.
   - Passing the property as a system property when starting the server. For example, `./startWebLogic.sh -Dweblogic.security.SSL.hostnameVerifier=weblogic.security.utils.SSLWLSWildcardHostnameVerifier`
   - Adding the property in the `EXTRA_JAVA_PROPERTIES` section of the *`DOMAIN_HOME`*`/bin/setDomainEnv.sh` script as:

–
`Dweblogic.security.SSL.hostnameVerifier=weblogic.security.utils.SSLWLSWildcardHostnameVerifier`

Consult your Oracle Identity Cloud Service administrator for any additional configuration requirements.

For detailed information about configuring TLS/SSL in WebLogic Server, see Configuring SSL. For information about using the WebLogic Remote Console to configure keystores and enable SSL, see the following topics in the *Oracle WebLogic Remote Console Online Help*:

- Identity and Trust
- Set Up TLS

Using the Oracle Identity Cloud Integrator Provider in FIPS Mode

In WebLogic Server 12.2.1.3 and earlier, when you enable FIPS mode for WebLogic Server and have configured the Oracle Identity Cloud Integrator provider, Java SSL Context initialization exceptions may occur, or users from Oracle Identity Cloud Service may fail to authenticate. These issues may be the result of your system using the default Java system truststore where the CA certificates store, `cacerts`, is not FIPS compliant.

When you enable JDK debug (`-Djavax.net.debug=ssl`), error messages for the exception are similar to the following:

```
Default SSLContext initialization
Key Store:
Key Store type: jks
Initializing key managers
Exception while initializing default context JKS keystores cannot be loaded
in FIPS-140 mode.
Only PKCS12 PBES2 key stores are supported
```

If you are using a PKCS12 keystore that is not FIPS compliant (created with the keytool command using the Sun JSSE provider for example), you may also receive an error similar to the following when using the keytool command:

```
keytool error: java.lang.SecurityException: Algorithm not allowable in
FIPS140 mode: PBE/PKCS12/SHA1/RC2/CBC/40
```

To address these errors and allow WebLogic Server to operate successfully, you need to first convert the JDK keystore to a FIPS compliant PKCS12 keystore, and then set the Java system properties to update the Java default settings for the truststore used with the default SSL Context. For details, see Converting the Default JKS Keystore for FIPS Compliance.

Authorization and Remote User HTTP Header Support

The Oracle Identity Cloud Integrator provider supports Oracle Identity Cloud Service access tokens via the `Authorization` HTTP header, and users validated by the Oracle Identity Cloud Service via the `REMOTE_USER` HTTP header.

Topics in this section include:

- Enabling Authorization and REMOTE_USER Header Support: Main Steps

- Ordering of Identity Assertion Headers

Enabling Authorization and REMOTE_USER Header Support: Main Steps

The `Authorization` and `REMOTE_USER` HTTP headers are not enabled by default. The services must be protected by Oracle Identity Cloud Service to safely accept user information from access tokens and from HTTP headers that contain no proof or signing materials. Therefore, you must enable the support for these headers before they can be accepted.

> **Note:**
>
> Use caution before enabling the REMOTE_USER HTTP header. This header should only be sent by a trusted client. You cannot have any publicly accessible endpoints if `REMOTE_USER` is enabled on the Oracle Identity Cloud Integrator provider. When exposing both public and protected endpoints, then use of `REMOTE_USER` may leave applications and WebLogic Server open to security vulnerabilities.

Only Oracle Identity Cloud Service identity tokens, `idcs_user_assertion` and `Idcs_user_assertion`, active types are accepted by default. To enable `Authorization` and `REMOTE_USER` HTTP header support:

1. In WebLogic Remote Console, in the **Edit Tree**, go to **Security**, then **Realms**, then *myRealm*, then **Authentication Providers**, then *myOracleIdentityCloudIntegratorProvider*. On the **Common** tab, in the **Active Types** field, add the `Authorization` header or the `REMOTE_USER` header or both.

   You can also use WLST, as shown in the script example. Note that you can enable these headers individually as required by your environment.

   ```
   connect('','','t3://host:port')
   Please enter your username :
   Please enter your password :
   ...
   edit()
   startEdit()
   realm = cmo.getSecurityConfiguration().getDefaultRealm()
   atn = realm.lookupAuthenticationProvider('IdentityCloudServiceIntegrator')
   atn.setActiveTypes(["idcs_user_assertion","REMOTE_USER","Authorization"])
   activate()
   disconnect()
   exit()
   #
   # Restart WebLogic Server
   #
   ```

2. To ensure that the process ordering for the multiple token types is defined, set the precedence order on the RealmMBean to specify the ordering for the different HTTP headers. See Ordering of Identity Assertion Headers.

Ordering of Identity Assertion Headers

When an HTTP request is processed by the WebLogic Server container, there may be multiple matches that can be used for identity assertion. Headers passed to the Oracle Identity Cloud

Integrator provider may contain an Oracle Identity Cloud Service identity token, Oracle Identity Cloud Service access token, or `REMOTE_USER`. However, the provider can only consume one active token type at a time. As a result there is no way to provide a set of tokens that can be consumed with one call. Therefore, the WebLogic Server container must choose between multiple tokens to perform identity assertion.

If you have enabled the `REMOTE_USER` or `Authorization` active types, you also need to update the `IdentityAssertionHeaderNamePrecedence` property on the RealmMBean to set the precedence order for the different HTTP headers, otherwise it is undefined.

Table 18-1 highlights some basic use cases and examples for precedence ordering in each case.

Table 18-1 HTTP Header Precedence Ordering for Different Use Cases

Use Case	Precedence Ordering	Comments
HTTP request may contain all supported tokens/headers	`Authorization: Bearer` (access token) `idcs_user_assertion` (identity token) `REMOTE_USER`	Settting this ordering gives precedence to claims from Identity Cloud Service tokens first. If no Identity Cloud Service tokens are supplied, then authentication falls back to using only remote user information. Oracle Identity Cloud Service tokens are not provided when the Oracle Identity Cloud Service handles basic authentication and then sends the `REMOTE_USER` HTTP request to the server.
HTTP requests contain primarily access tokens and HTTP Basic authentication with some Web single sign-on (SSO) tokens	`Authorization: Bearer` (access token) `REMOTE_USER`	In some cases additional security context may be required, such as the scopes from the access token. When additional information is required, then access and identity tokens should take precedence over remote user information. Setting this ordering gives precedence to access tokens and establishes a security context that includes user, client, application roles, scopes and audience data from the access token. For Web SSO and HTTP basic credentials that have been verified by the Oracle Identity Cloud Service, the deployed application uses the remote user information established in the security context, including the user's Identity Cloud Service application roles. **Note:** When Oracle Identity Cloud Service identity tokens are included, the remote user information is still preferred and the user's Identity Cloud Service application roles are still obtained.

Table 18-1 (Cont.) HTTP Header Precedence Ordering for Different Use Cases

Use Case	Precedence Ordering	Comments
HTTP requests may contain multiple tokens but remote user should take precedence	`REMOTE_USER`	If the service wants to use just the remote user information, setting remote user as the highest precedence ensures that HTTP requests that contain `REMOTE_USER` are given precedence. User information in the other tokens is ignored. If remote user information is omitted, then authentication falls back to Identity Cloud Service tokens. **Note:** When using the `<auth-method>CLIENT-CERT, BASIC</auth-method>` to define an authentication mechanism for the application, the remote user information is still preferred over the Authorization: Basic credential because `CLIENT-CERT` is the first method used for authentication by the Web Container. Therefore, the HTTP BASIC credential is only processed if the assertion from the remote user information (tokens) fails or the token related HTTP headers are omitted from the HTTP request. See Providing a Fallback Mechanism for Authentication Methods in *Developing Applications with the WebLogic Security Service*.

Setting HTTP Header Precedence: WLST Example

You can use WLST online to set the HTTP Token precedence order as shown here. This example sets the ordering to authorization (access token), identity token, and then remote user as shown in the first use case.

```
connect('','','t3://host:port')
Please enter your username :
Please enter your password :
...
edit()
startEdit()
realm = cmo.getSecurityConfiguration().getDefaultRealm()
realm.setIdentityAssertionHeaderNamePrecedence(["Authorization:
Bearer","idcs_user_assertion","REMOTE_USER"])
activate()
disconnect()
exit()
#
# It is not necessary to restart the server.
#
```

Handling Authentication Failures

Authentication failures occur if the Oracle Identity Cloud Service is unavailable, or not responding to authentication requests. When these failures occur, you can modify the settings on the Oracle Identity Cloud Integrator provider to control how the authenticator handles the failures.

When the Oracle Identity Cloud Service is unavailable, the authentication failures are logged to the server log at periodic intervals. You can specify the interval at which the count of the authentication failures is logged to the server log using the `ServerNotAvailableCounterInterval` configuration attribute. By default, the failures are logged every five minutes. If you set the value of the attribute to zero or a negative value, the count of failures is not logged.

When the Oracle Identity Cloud Service is not responding to authentication requests and returns a Too Many Requests error, you can use the `ServerBackoffEnabled` configuration attribute to specify whether the server should back off and retry the request. By default, this attribute is set to `true`.

You can set these properties on the Provider Specific page in WebLogic Remote Console, using WLST, or directly on the MBean.

19

Configuring the WebLogic OpenID Connect Provider

The WebLogic OpenID Connect provider is an authentication and identity assertion provider that facilitates access to web applications from users and groups stored in external authorization servers that adhere to the OpenID Connect and OAuth 2.0 standards.

Topics in this section include:

About the WebLogic OpenID Connect Provider

The WebLogic OpenID Connect provider combines authentication and identity assertion into a single provider. The provider establishes identity (the Subject) on WebLogic Server with the authenticated user and the user's groups when the identity store is a supported OpenID provider.

The WebLogic OpenID Connect provider consumes ID tokens for authenticating to applications and uses them to establish authenticated subjects.

> **Note:**
>
> It is important to distinguish OpenID providers from the WebLogic OpenID *Connect* provider. OpenID providers are external authorization servers that adhere to OAuth 2.0 and OpenID Connect standards and provide authentication as a service. The WebLogic OpenID Connect provider is a WebLogic security provider, a module that integrates with the security realm to add support for authentication and identity assertion services.

The WebLogic OpenID Connect provider is controlled by `OIDCIdentityAsserterMBean` and includes the following configuration attributes:

- `ClockScrew`
- `KeyCacheSize`
- `KeyCacheTTL`
- `RequestCacheSize`
- `RequestCacheTTL`
- `UserIDTokenClaim`
- `UserNameTokenClaim`

- `VirtualUserAllowed`

For information on these attributes, see OIDCIdentityAsserterMBean in *MBean Reference for Oracle WebLogic Server*.

The WebLogic OpenID Connect provider currently supports the following OpenID providers:

- Keycloak
- Microsoft Azure

Configure the WebLogic OpenID Connect Identity Assertion Provider in WebLogic Remote Console

The WebLogic OpenID Connect provider is an authentication and identity assertion provider that delegates authentication services for web applications to OpenID providers.

1. In WebLogic Remote Console, expand the **Edit Tree** and go to **Security**, then **Realms**, then *myRealm*, then **Authentication Providers**.
2. Click **New**.
3. Enter a name for the new provider in the **Name** field.
4. From the **Type** drop-down list, select the **WebLogic OpenID Connect Identity Asserter** provider.
5. Click **Create**.
6. On the **Common** tab, update any attributes applicable to your environment and click **Save**.
7. On the **OIDC Identity Asserter Parameters** tab, update any attributes applicable to your environment.
8. Click **Save**.
9. If you are using the Default Authenticator provider (WebLogic Authentication provider), then you must set the JAAS Control Flag option on the Default Authenticator to `SUFFICIENT`. See Set the JAAS Control Flag in *Oracle WebLogic Remote Console Online Help*.

   For a description of the JAAS control flag and how multiple authentication providers interact in a domain, see Setting the JAAS Control Flag Option and Using More Than One Authentication Provider.

You must perform some additional configuration in web applications before they can use OpenID providers for authentication. See Preparing Web Applications for the WebLogic OpenID Connect Provider.

Preparing Web Applications for the WebLogic OpenID Connect Provider

Before you can use the WebLogic OpenID Connect provider to delegate authentication of web applications to an OpenID provider, you must create an OAuth client in the OpenID provider and then include certain client attributes in the web application.

You must create an `oidcAuth.properties` file that contains the configuration information that WebLogic Server uses to determine which OpenID provider is responsible for providing authenticating services for the application.

1. In the `WEB-INF/` directory of the web application, create a new file and save it as `oidcAuth.properties`.
2. Retrieve the following configuration attributes from the OAuth client in the OpenID provider and add them to `oidcAuth.properties` in the following format:

   ```
   issuer=issuer identifier
   clientId=client ID
   clientSecret=client secret
   redirectUrl=redirect URL
   ```

   For example:

   ```
   issuer=https://example.com:8443:/realms/dev
   clientId=devclient
   clientSecret=57gw6LVlkWUTWDmbksiwH96ihFgpbF6d8
   redirectUrl=https://organization.com:7002/devapp/go
   ```

3. Save your changes.
4. Continue with your application development and deployment process.

> **Note:**
>
> Whenever you make any changes to `oidcAuth.properties`, you must repackage your application and re-deploy it to implement those changes.

Part IV
Configuring Single Sign-On

Learn how to configure the various single sign-on solutions available for Oracle WebLogic Server.

This part contains the following chapters:

20

Configuring Single Sign-On with Microsoft Clients

Learn how to set up single sign-on (SSO) between Oracle WebLogic Server and Microsoft clients, using Windows Integrated Authentication based on the Simple and Protected Negotiate (SPNEGO) mechanism and the Kerberos protocol, together with the WebLogic Negotiate Identity Assertion provider.

Overview of Single Sign-On with Microsoft Clients

Single sign-on (SSO) with Microsoft clients allows cross-platform authentication between Web applications or Web services running in a WebLogic domain and .NET Web service clients or browser clients (for example, Internet Explorer) in a Microsoft domain.

The Microsoft clients must use Windows Integrated Authentication based on the Simple and Protected Negotiate (SPNEGO) mechanism. Cross-platform authentication is achieved by emulating the negotiate behavior of native Windows-to-Windows authentication services that use the Kerberos protocol. In order for cross-platform authentication to work, non-Windows servers (in this case, WebLogic Server) need to parse SPNEGO tokens in order to extract Kerberos tokens which are then used for authentication.

Refer to the Microsoft documentation for details about Kerberos authentication support on Windows.

> **Note:**
>
> WebLogic Server's Single sign-on (SSO) support for Microsoft clients is available only for Web applications and not for other application types, such as EJBs.

System Requirements for SSO with Microsoft Clients

To use SSO with Microsoft clients, you must meet both host computer requirements and client computer requirements.

Host Computer Requirements for Supporting SSO with Microsoft Clients

The host computer that supports SSO for Microsoft clients must meet the following system requirements:

- A version of Microsoft Windows that is supported by WebLogic Server for SSO with Microsoft clients

  For information about these supported versions, see Oracle Fusion Middleware Supported System Configurations.
- Fully-configured Active Directory authentication service. Specific Active Directory requirements include:
  - User accounts for mapping Kerberos services
  - Service Principal Names (SPNs) for those accounts
  - Keytab files created and copied to the start-up directory in the WebLogic domain
- WebLogic Server installed and configured properly to authenticate through Kerberos, as described in this chapter

Oracle recommends encrypting the user accounts that are mapped to Kerberos services on the WebLogic Server host. However, the ability to encrypt these accounts imposes additional requirements. The specific requirements depend on the encryption algorithm used, as shown in Table 20-1. For each encryption algorithm listed in Table 20-1, see the Oracle Fusion Middleware Supported System Configurations page for information about:

- The corresponding version of Microsoft Windows that is supported as the Active Directory platform.
- The specific versions of the Internet Explorer and Mozilla FireFox client types that are supported.

Table 20-1 Client Type Requirements for Using Encrypted User Accounts

Encryption Algorithm	Supported Client Type
DES	• Internet Explorer • Mozilla FireFox • .NET Web service • Java SE client
AES-128, AES-256, and RC4	• Internet Explorer • Mozilla FireFox • Java SE client[1]

[1] User accounts accessed with a Java SE client can also be encrypted with DES, AES-128, AES-256, and RC4 and defined in Active Directory running on a Microsoft Windows platform supported by WebLogic Server for this purpose.

Client Computer Requirements for Supporting Microsoft Clients Using SSO

The computer hosting a Microsoft client that uses SSO must meet the following requirements:

- An installation of Microsoft Windows
- Include one of the client types listed in the following table, which also includes links to the instructions for configuring those clients:

Table 20-2 Configuration of Supported Hosted Clients

For the following client type . . .	See the following topic . . .
Internet Explorer[1]	Configuring an Internet Explorer Browser
Mozilla FireFox [1]	Configuring a Mozilla Firefox Browser
.NET Framework with properly configured web service client	Configuring a .NET Web Service
Standalone Java SE client application	Configuring a Java SE Client Application

[1] For information about the specific version supported for accessing user accounts that are defined in Active Directory and encrypted with AES-128, AES-256, or RC4, see Oracle Fusion Middleware Supported System Configurations.

Clients must be logged on to a Microsoft Windows domain and have Kerberos credentials acquired from the Active Directory server in the domain. Local logins are not supported.

> **Note:**
>
> For information about the versions of Microsoft Windows that are supported for hosting clients using SSO, and the encryption algorithms with which user accounts accessed by those clients can be defined in Active Directory, see Oracle Fusion Middleware Supported System Configurations.

Single Sign-On with Microsoft Clients: Main Steps

Configuring SSO with Microsoft clients requires set-up procedures in the Microsoft Active Directory, the client, and the WebLogic domain.

The procedure for configuring SSO with Microsoft client are detailed in the sections that follow.

- Define a principal in Active Directory to represent the WebLogic Server. The Kerberos protocol uses the Active Directory server in the Microsoft domain to store the necessary security information.
- Any Microsoft client you want to access in the Microsoft domain must be set up to use Windows Integrated Authentication, sending a Kerberos ticket when available.
- In the security realm of the WebLogic domain, configure a Negotiate Identity Assertion provider. The Web application or Web service used in SSO needs to have authentication set in a specific manner. A JAAS login file that defines the location of the Kerberos identification for WebLogic Server must be created.

To configure SSO with Microsoft clients:

1. Configure your network domain to use Kerberos. See Configuring Your Network Domain to Use Kerberos.
2. Create a Kerberos identification for WebLogic Server.
   a. Create a user account in the Active Directory for the host on which WebLogic Server is running.
   b. Create a service principal name (SPN) for this account.
   c. Create a user mapping and keytab file for this account.

   See Creating a Kerberos Identification for WebLogic Server.
3. Choose a Microsoft client (either a Web service or browser) or a Java SE client and configure it to use Windows Integrated Authentication. See Configuring Microsoft Clients to Use Windows Integrated Authentication.
4. Set up the WebLogic domain to use Kerberos authentication.
   a. Create a JAAS login file that points to the Active Directory server in the Microsoft domain and the keytab file created in Step 1. See Creating a JAAS Login File.
   b. Configure a Negotiate Identity Assertion provider in the WebLogic Server security realm. See Configuring a Negotiate Identity Assertion Provider.
5. Start WebLogic Server using specific start-up arguments. See Using Startup Arguments for Kerberos Authentication with WebLogic Server.

The following sections describe these steps in detail.

Configuring Your Network Domain to Use Kerberos

To configure Kerberos in your Windows domain controller, you need to configure each machine that will access the Key Distribution Center (KDC) to locate the Kerberos realm and available KDC servers.

A Windows domain controller can serve as the Kerberos Key Distribution Center (KDC) server for Kerberos-based client and host systems. On any domain controller, the Active Directory and the Kerberos services are running automatically.

Java GSS requires a Kerberos configuration file. The default name and location of the Kerberos configuration file depends on the operating system being used. Java GSS uses the following order to search for the default configuration file:

1. The file referenced by the Java property `java.security.krb5.conf`.
2. `${java.home}/lib/security/krb5.conf`.
3. `%windir%\krb5.ini` on Microsoft Windows platforms.
4. `/etc/krb5/krb5.conf` on Solaris platforms.
5. `/etc/krb5.conf` on other UNIX platforms.

For example:

Example 20-1 Sample krb5.ini File

```
[libdefaults]
default_realm = EXAMPLE.COM (Identifies the default realm. Set its value to your
Kerberos realm)
default_tkt_enctypes = des-cbc-crc
```

```
default_tgs_enctypes = des-cbc-crc
ticket_lifetime = 600

[realms]
EXAMPLE.COM = {
kdc = <IP address for MachineA> (host running the KDC)
(For UNIX systems, you need to specify port 88, as in <IP-address>:88)
admin_server = MachineA
default_domain = EXAMPLE.COM
}

[domain_realm]
.example.com = EXAMPLE.COM

[appdefaults]
autologin = true
forward = true
forwardable = true
encrypt = true
```

Creating a Kerberos Identification for WebLogic Server

Active Directory provides support for service principal names (SPN), which are a key component in Kerberos authentication. You must define an SPN to represent your WebLogic Server in the Kerberos realm. Learn how to create an SPN, user mapping, and keytab file for WebLogic Server.

SPNs are unique identifiers for services running on servers. Every service that uses Kerberos authentication needs to have an SPN set for it so that clients can identify the service on the network. An SPN usually looks something like *name@YOUR.REALM*. If an SPN is not set for a service, clients have no way of locating that service. Without correctly set SPNs, Kerberos authentication is not possible. Keytab files are the mechanism for storing the SPNs. Keytab files are copied to the WebLogic domain and are used in the login process. This configuration step describes how to create an SPN, user mapping, and keytab file for WebLogic Server.

This configuration process requires the use of the following Active Directory utilities:

- `setspn`—Microsoft Windows Resource Kit
- `ktpass`—Microsoft Windows distribution CD in `Program Files\Support Tools`

> **Note:**
>
> The `setspn` and `ktpass` Active Directory utilities are products of Microsoft. Therefore, Oracle does not provide complete documentation for this utilities. See the appropriate Microsoft documentation for more information.

The process for creating a Kerberos identification consists of the following steps:

- Step 1: Create a User Account for the Host Computer
- Step 2: Configure the User Account to Comply with Kerberos
- Step 3: Define a Service Principal Name and Create a Keytab for the Service
- Step 4: Verify Correct Setup

Step 1: Create a User Account for the Host Computer

In the Active Directory server, create a user account for the host computer on which WebLogic Server runs. (Select **New** > **User**, not **New** > **Machine**.)

When creating the user account, use a unique name to represent the host computer on which WebLogic Server runs. **If your WebLogic Server instance runs on a host that is part of the Active Directory domain, then you must use a name other than the host name**. For example, if the host is named `myhost.example.com`, create a user in Active Directory called `myweblogichost`. If your WebLogic Server instance runs on a machine that is not a part of the Active Directory domain, then you may use any unique name (including the host name) for creating the user account.

Note the password you defined when creating the user account. You will need it for the instructions described in Step 3: Define a Service Principal Name and Create a Keytab for the Service. Do not select the **User must change password at next logon** option or any other password options.

Step 2: Configure the User Account to Comply with Kerberos

Configure the new user account to comply with the Kerberos protocol as follows. The user account's encryption type must be DES, at a minimum, and the account must require Kerberos pre-authentication. Stronger encryption types are supported, including AES-128, AES-256, and RC4.

> **Note:**
>
> The use of a particular encryption type has a dependency on the version of the Microsoft Windows platform on which Active Directory runs. For more information, including a list of supported encryption types, see Oracle Fusion Middleware Supported System Configurations.

1. Right-click the name of the user account in the Users tree in the left pane and select **Properties**.
2. Select the **Account** tab and check the following:
   - If you plan to use DES encryption, check the box **Use DES encryption types for this account**.
   - If you plan to use AES encryption, check the boxes **This account supports Kerberos AES 128** and **This account supports Kerberos AES 256**, and make sure that **Use Kerberos DES Encryption** is unchecked.

   Make sure no other boxes are checked, particularly the box "Do not require Kerberos pre-authentication."
3. Click **OK**.

> **Note:**
>
> Setting the encryption type may corrupt the password. Therefore, reset the user password by right-clicking the name of the user account, selecting **Reset Password**, and re-entering the password created in Step 1: Create a User Account for the Host Computer.

Step 3: Define a Service Principal Name and Create a Keytab for the Service

As mentioned in Creating a Kerberos Identification for WebLogic Server, an SPN is a unique name that identifies an instance of a service and is associated with the logon account under which the service instance runs. The SPN is used in the process of mutual authentication between the client and the server hosting a particular service. The client finds a computer account based on the SPN of the service to which it is trying to connect. So, in a specific project, you need to link the service that will be invoked by your WebLogic clients to the account you just defined for your WebLogic Server. For example, the service invoked by the WebLogic browser clients is `HTTP/myhost.example.com`, which needs to be linked to the `myhost` account.

Windows account names are not multipart as Kerberos principal names. Because of this, it is not possible to directly create an account using the name `HTTP/`*`hostname.dns.com`*. Such a principal instance is created through SPN mappings. In this case, an account is created with a meaningful name *`hostname`*, and an SPN mapping is added for `HTTP/`*`hostname.dns.com`*.

The specific steps for defining an SPN and creating a keytab for the service depend on the underlying platform on which WebLogic Server is running. They are provided in the following sections:

Defining an SPN and Creating a Keytab on Windows Systems

If WebLogic Server runs on a Windows system, complete the following steps:

1. Use the `setspn` utility to create the SPN for the HTTP service for the WebLogic Server account created in Step 1. For example:

   ```
   setspn -A HTTP/myhost.example.com myhost
   ```

2. Identify the SPNs that are associated with your user account by entering the `setspn -L` command. For example:

   ```
   setspn -L myhost
   ```

   > **Note:**
   >
   > The preceding is an important step. If the same service is linked to a different account in the Active Directory server, the client will not send a Kerberos ticket to the server.

3. Use the `ktab` utility to create a keytab to be exported to the WebLogic Server machine. The command to run the `ktab` utility has the following syntax (note that the Kerberos realm name must be entered in all uppercase):

```
ktab -k keytab-file-name -a account-name@REALM.NAME
```

For example:

```
ktab -k mykeytab -a myhost@EXAMPLE.COM
```

When prompted for a password, enter the password created in Step 1: Create a User Account for the Host Computer.

4. Save the keytab file in a secure location and export it to the domain directory of your WebLogic Server instance (for example, to `myhost`).

Defining an SPN and Creating a Keytab on UNIX Systems

If WebLogic Server runs on a UNIX system, create a service principal name (SPN) and a keytab file for the HTTP service for the WebLogic Server account by using the `ktpass` command-line tool. This tool enables an administrator to configure a non-Windows Server Kerberos service as a security principal in the Windows Server Active Directory.

The `ktpass` command configures the SPN for the service in Active Directory and generates a Kerberos keytab file containing the shared secret key of the service. The tool allows UNIX-based services that support Kerberos authentication to use the interoperability features provided by the Windows Server Kerberos KDC service.

The `ktpass` command has the following syntax:

```
ktpass -princ HTTP/hostname@REALM-NAME -mapuser account-name -pass password -out keytab-file-name -crypto algorithm -ptype KRB5_NT_PRINCIPAL
```

In the preceding syntax, *`algorithm`* identifies the encryption algorithm to use. If you are using AES, specify `AES128-SHA1` or `AES256-SHA1`. For example:

```
ktpass -princ HTTP/myhost.example.com@EXAMPLE.COM -mapuser myhost -pass password -out mykeytab -crypto AES256-SHA1 -ptype KRB5_NT_PRINCIPAL
```

> **Note:**
>
> On UNIX systems, creating an SPN that uses a DES or an AES encryption algorithm is supported as of JDK 1.6.0_24 or later.

To verify that the SPN and the keytab file are set up correctly, you can do the following:

- Use the `setspn -l` command to verify that the SPN is mapped successfully.
- Use the klist command to show `Key type: 17` for AES-128, and `Key type: 18` for AES-256. For example:

  ```
  -klist -e -k keytab-file-name
  ```

- Use the kinit command to verify that the Kerberos setup and keytab are valid.

> **Note:**
>
> The `ktpass` command changes the principal name in the Active Directory server from *`account-name`* to `HTTP/`*`account-name`*. Consequently, the keytab file is generated for a principal named `HTTP/`*`account-name`*. However, sometimes the name change does not happen. If not, you should change it manually in the Active Directory server; otherwise the keytab you generate will not work properly.
>
> To modify the user logon name manually:
>
> 1. Right-click on the user node, select **Properties**, and click on the **Account** tab.
> 2. Export the generated keytab file to your UNIX machine hosting the WebLogic Server in the WebLogic domain directory.

Step 4: Verify Correct Setup

You can use the following utilities to verify that your SPN and keytab files are set up correctly.

- Use the `setspn -l` command to verify that the SPN is mapped successfully.
- Use the `klist` command to verify the key type. For example:

  ```
  -klist -e -k keytab-file-name
  ```

  For AES 128, this command displays `Key type: 17`. For AES 256, `Key type: 18` is displayed.
- Use the `kinit` utility to verify that Kerberos is set up properly and that your principal and keytab are valid.

  The `kinit` utility is provided by the JRE and may be used to obtain and cache Kerberos ticket-granting tickets. You can run the `kinit` utility by entering the following command:

  ```
  kinit -k -t keytab-file account-name
  ```

  The output should appear similar to the following:

  ```
  New ticket is stored in cache file C:\Documents and Settings\Username\krb5cc_myhost
  ```

Configuring Microsoft Clients to Use Windows Integrated Authentication

You must ensure that the Microsoft client you want to use for single sign-on is configured to use Windows Integrated Authentication. You can configure a .NET Web server, an Internet Explorer browser client, a Mozilla Firefox client, or a Java SE client to use Windows Integrated Authentication.

This section contains the following topics:

> **Note:**
>
> If the SPN for the user account on the WebLogic Server host to which the Kerberos ticket is mapped is configured to use DES or AES-256 encryption (see Step 2: Configure the User Account to Comply with Kerberos), the client must be running with a supported JDK. See Oracle Fusion Middleware Supported System Configurations.

Configuring a .NET Web Service

To configure a .NET Web service to use Windows Integrated Authentication:

1. In the `web.config` file for the Web service, set the authentication mode to Windows for IIS and ASP.NET as follows:

   ```
   <authentication mode="Windows" />
   ```

   This setting is usually the default.

2. Add the statement needed for the Web services client to pass to the proxy Web service object so that the credentials are sent through SOAP.

   For example, if you have a Web service client for a Web service that is represented by the proxy object `conv`, the syntax is as follows:

   ```
   /*
   * Explicitly pass credentials to the Web Service
   */
   conv.Credentials =
   System.Net.CredentialCache.DefaultCredentials;
   ```

Configuring an Internet Explorer Browser

To configure an Internet Explorer browser to use Windows Integrated Authentication, complete the procedures described in the following sections:

Configure Local Intranet Domains

In Internet Explorer:

1. Select **Tools** > **Internet Options**.
2. Select the Security tab.
3. Select **Local intranet** and click **Sites**.
4. In the Local intranet popup, ensure that the "Include all sites that bypass the proxy server" and "Include all local (intranet) sites not listed in other zones" options are checked.
5. Click **Advanced**.
6. In the Local intranet (Advanced) dialog box, add all relative domain names that will be used for WebLogic Server instances participating in the SSO configuration (for example, `myhost.example.com`) and click OK.

Configure Intranet Authentication

In Internet Explorer:

1. Select **Tools** > **Internet Options**.
2. Select the Security tab.
3. Select **Local intranet** and click **Custom Level...** .
4. In the Security Settings dialog box, scroll to the User Authentication section.
5. Select **Automatic logon only in Intranet zone**. This option prevents users from having to re-enter logon credentials, which is a key piece to this solution.
6. Click OK.

Verify the Proxy Settings

If you have a proxy server enabled:

1. In Internet Explorer, select **Tools** > **Internet Options**.
2. Select the Connections tab and click **LAN Settings**.
3. Verify that the proxy server address and port number are correct.
4. Click **Advanced**.
5. In the Proxy Settings dialog box, ensure that all desired domain names are entered in the Exceptions field.
6. Click **OK** to close the Proxy Settings dialog box.

Set Integrated Authentication for Older Internet Explorer Versions

If you are configuring an older version of Internet Explorer, you might also need to complete the following steps:

1. In Internet Explorer, select **Tools** > **Internet Options**.
2. Select the Advanced tab.
3. Scroll to the Security section.
4. Verify that the Enable Integrated Windows Integrated Authentication option is checked and click **OK**.

   If this option was not checked, check it, click **OK**, and restart the computer.

Configuring a Mozilla Firefox Browser

To configure a Firefox browser to use Windows Integrated Authentication, complete the following steps:

1. Start Firefox.
2. In the Location Bar, enter `about:config`.
3. Enter the filter string `network.negotiate`.
4. Set the preferences as shown in Table 20-3.

Table 20-3 Preferences Required in Firefox for Windows Integrated Authentication

Preference Name	Status	Type	Value
network.negotiate-auth.allow-proxies	default	boolean	true
network.negotiate-auth.delegation-uris	user set	string	http://,https://
network.negotiate-auth.gsslib	default	string	<blank>[1]
network.negotiate-auth.trusted-uris	user set	string	http://,https://
network.negotiate-auth.using-native-gsslib	default	boolean	true

[1] The value for the network.negotiate-auth.gsslib preference should be kept blank.

Configuring a Java SE Client Application

To configure a Java SE client application to use Windows Integrated Authentication, complete the following steps:

1. Ensure that your Java SE client is running with a supported JDK. See Oracle Fusion Middleware Supported System Configurations.
2. Create a JAAS configuration file that identifies the Kerberos login module, com.sun.security.auth.module.Krb5LoginModule. This configuration file defines two login entries:
   - com.sun.security.jgss.krb5.initiate — Invoked for the Java client.
   - com.sun.security.jgss.krb5.accept — Invoked for the WebLogic Server instance that is represented by a Kerberos identity and that hosts the Web application to which the client wants access.

   For each login entry, options are included that:
   - Require that authentication of the principal must succeed (that is, the user of the client application who is defined in the Microsoft domain).
   - Set useKeyTab to true, which causes the principal's key to be obtained from the keytab.
   - Identify the name of the keytab.
   - Set storeKey to true, which causes the principal's key to be stored in the Subject.
   - Optionally, set the debug option to true.

   The following example shows JAAS configuration file for the Kerberos login module used for the principal negotiatetester, who is defined in the Microsoft domain, SECURITYQA.COM, in which the Active Directory server runs:

```
com.sun.security.jgss.krb5.initiate {
   com.sun.security.auth.module.Krb5LoginModule
   required principal="negotiatetester@SECURITYQA.COM"
   useKeyTab=true
   keyTab=negotiatetester_keytab storeKey=true debug=true; };

com.sun.security.jgss.krb5.accept {
   com.sun.security.auth.module.Krb5LoginModule
```

```
    required principal="negotiatetester@SECURITYQA.COM"
    useKeyTab=true keyTab=negotiatetester_keytab storeKey=true debug=true; };
```

3. In the Java command that starts the client application, pass the following values as arguments:
   - The Microsoft domain in which the Active Directory server runs
   - The host name of the computer running the Kerberos Key Distribution Center (KDC) server
   - The JAAS configuration file that identifies the Kerberos login module
   - The `javax.security.auth.useSubjectCredsOnly=false` property, which specifies that it is permissible to use an authentication mechanism other than Subject credentials
   - The name of the Java SE client class
   - The Web application resource to which access is requested

   For example:

```
java -Djava.security.krb5.realm = SECURITYQA.COM\
-Djava.security.krb5.kdc = rno05089.example.com\
-Djava.security.auth.login.config = negotiatetester_krb5Login.conf\
-Djavax.security.auth.useSubjectCredsOnly = false\
RunHttpSpnego http://myhost.example.com:7001/AuthenticatedServlet.jsp
```

Creating a JAAS Login File

If you are running WebLogic Server on either the Windows or UNIX platforms, you must create a JAAS login file. You must correctly specify the values of the `userPrincipalName` attribute and the `keytab` option in the JAAS login file.

The JAAS login file tells the WebLogic Security Framework to use Kerberos authentication and defines the location of the keytab file which contains Kerberos identification information for WebLogic Server. You specify the location of the JAAS login file in the `java.security.auth.login.config` startup argument for WebLogic Server, as described in Using Startup Arguments for Kerberos Authentication with WebLogic Server.

> **Note:**
>
> The JAAS Login Entry names are `com.sun.security.jgss.krb5.initiate` and `com.sun.security.jgss.krb5.accept`.

The following example shows a sample JAAS login file for Kerberos authentication. Significant sections are shown in **bold**.

Example 20-2 Sample JAAS Login File for Kerberos Authentication

```
com.sun.security.jgss.krb5.initiate {

     com.sun.security.auth.module.Krb5LoginModule required
     principal="myhost@Example.CORP" useKeyTab="true"
     keyTab="mykeytab" storeKey="true";
};

com.sun.security.jgss.krb5.accept {
```

```
      com.sun.security.auth.module.Krb5LoginModule required
      principal="myhost@Example.CORP" useKeyTab="true"
      keyTab="mykeytab" storeKey="true";

};
```

For the principal option, specify the value of the `userPrincipalName` attribute of the account under which the service is running. (Incorrectly specifying the user principal name results in an error such as "Unable to obtain password from user.")

The keytab file specified in the `keytab` option must be accessible by the WebLogic Server process. Ensure that the appropriate permissions are set. If you are unsure of the search path WebLogic Server is using, provide the absolute path to the file. Make sure that you enclose the path in double quotes, and replace any backslash (`\`) in the path with a double backslash (`\\`) or a forward slash (`/`).

Configuring the Identity Assertion Provider

The Negotiate Identity Assertion provider decodes Simple and Protected Negotiate (SPNEGO) tokens to obtain Kerberos tokens, validates the Kerberos tokens, and maps Kerberos tokens to WebLogic users.

You need to configure a Negotiate Identity Assertion provider in your WebLogic security realm in order to enable SSO with Microsoft clients. See Configuring a Negotiate Identity Assertion Provider in this document, and also see Configure an Authentication or Identity Assertion Provider in *Oracle WebLogic Remote Console Online Help*.

Using Startup Arguments for Kerberos Authentication with WebLogic Server

Startup arguments are used for authenticating Kerberos with WebLogic Server.
To use Kerberos authentication with WebLogic Server, use the following arguments in the Java command to start WebLogic Server:

```
-Djavax.security.auth.useSubjectCredsOnly=false
-Djava.security.auth.login.config=krb5Login.conf
-Djava.security.krb5.realm=Example.CORP
-Djava.security.krb5.kdc=ADhostname
```

In the preceding list of arguments:

- `javax.security.auth.useSubjectCredsOnly` specifies that it is permissible to use an authentication mechanism other than Subject credentials.
- `java.security.auth.login.config` specifies the JAAS login file, `krb5Login.conf`, described in Creating a JAAS Login File.
- `java.security.krb5.realm` defines the Microsoft domain in which the Active Directory server runs.
- `java.security.krb5.kdc` defines the host name on which the Active Directory server runs.

Java GSS messages are often very useful during troubleshooting, so you might want to add `-Dsun.security.krb5.debug=true` as part of the initial setup.

Verifying Configuration of SSO with Microsoft Clients

To verify that SSO with Microsoft clients is configured properly, point a browser to the Microsoft Web application or Web service you want to use.

For the verification to work properly, the browser must be configured as described in Configuring an Internet Explorer Browser. If you are logged on to a Windows domain and have Kerberos credentials acquired from the Active Directory server in the domain, you should be able to access the Web application or Web service without providing a username or password.

21

Configuring Single Sign-On with Web Browsers and HTTP Clients Using SAML

WebLogic Server supports single sign-on (SSO) based on SAML. You configure single sign-on with Web browsers or other HTTP clients by using authentication based on the Security Assertion Markup Language (SAML) versions 1.1 and 2.0.

> **Note:**
>
> The SAML 1.1 Identity Assertion provider, the SAML 1.1 credential mapping provider, and related configuration and services for SAML 1.1 federation services are deprecated as of WebLogic Server 14.1.2.0.0 and will be removed in a future release. Oracle recommends using SAML 2.0.

SAML enables cross-platform authentication between Web applications or Web services running in an Oracle WebLogic Server domain and Web browsers or other HTTP clients. When users are authenticated at one site that participates in a single sign-on (SSO) configuration, they are automatically authenticated at other sites in the SSO configuration and do not need to log in separately.

> **Note:**
>
> - A WebLogic Server instance that is configured for SAML 2.0 SSO cannot send a request to a server instance configured for SAML 1.1, and vice-versa.
> - WebLogic Server supports encrypted SAML assertions for SAML 2.0.
> - WebLogic Server supports SAML Single Logout for the WebLogic SAML Service Provider.

For an overview of SAML-based single sign on, see the following topics in *Understanding Security for Oracle WebLogic Server*:

- Security Assertion Markup Language (SAML)
- Web Browsers and HTTP Clients via SAML
- Single Sign-On with the WebLogic Security Framework

This chapter includes the following sections:

Configuring SAML Services

The way to configure SAML services for single sign-on with Web browsers and HTTP clients depends on the specific version of SAML you plan to use.

Refer to the following table:

To configure the following version of SAML . . .	See the following chapter . . .
SAML 1.1	Configuring SAML 1.1 Services
SAML 2.0	Configuring SAML 2.0 Services

Configuring Single Sign-On Using SAML White Paper

The *Configuring Single Sign-On using SAML in WebLogic Server 9.2* white paper provides step-by-step instructions for configuring the single sign-on capability between two simple Java EE Web applications running on two different WebLogic domains.

The SAML configuration for single sign-on that is described in the *Configuring Single Sign-On using SAML in WebLogic Server 9.2* white paper (`http://www.oracle.com/technetwork/articles/entarch/sso-with-saml-099684.html`) is performed using the WebLogic Server 9.2 Administration Console with no programming involved. The tutorial also briefly introduces the basic interactions between WebLogic containers, the security providers, and the security framework during the single sign-on process.

Although it is based on a previous version of WebLogic Server, you may find this tutorial to be a useful resource as you develop your own SAML implementation.

SAML for Web Single Sign-On Scenario API Example

When you install the Server Examples component of WebLogic Server, which is available by performing a custom installation, WebLogic Server installs several API code examples. Included among the security API examples is SAML for Web single sign-on (SSO) Scenario.

The Web SSO example, which you build, run, and deploy, shows a variety of SSO configurations for your applications using WebLogic Server and SAML. The Server Examples provide access to code examples and sample applications that offer several approaches to learning about and working with WebLogic Server.

The following three scenarios are included:

- SAML 2.0 POST binding
- SAML 1.1
- SAML 2.0 Artifact binding with custom attributes

All files needed to build, deploy, and run the example are included, as are the scripts that configure the WebLogic domains that are used. For more information about the examples, including the directories in which they are installed, see Sample Application and Code Examples in *Understanding Oracle WebLogic Server*.

22

Configuring SAML 1.1 Services

> **Note:**
>
> The SAML 1.1 Identity Assertion provider, the SAML 1.1 Credential Mapping provider, and related configuration and services for SAML 1.1 federation services are deprecated as of WebLogic Server 14.1.2.0.0 and will be removed in a future release. Oracle recommends using SAML 2.0.

Learn how to configure single sign-on in Oracle WebLogic Server with Web browsers and HTTP clients using SAML 1.1.

In addition to the topics described in these sections, see Creating Assertions for Non-WebLogic SAML 1.1 Relying Parties in *Developing Applications with the WebLogic Security Service* for information on how to create a custom SAML name mapper that maps Subjects to specific SAML 1.1 assertion attributes required by a third-party SAML Relying Party.

This chapter includes the following sections:

Enabling Single Sign-on with SAML 1.1: Main Steps

To enable single sign-on with SAML, configure WebLogic Server as either a source site or destination site.

> **Note:**
>
> The SAML 1.1 Identity Assertion provider, the SAML 1.1 Credential Mapping provider, and related configuration and services for SAML 1.1 federation services are deprecated as of WebLogic Server 14.1.2.0.0 and will be removed in a future release. Oracle recommends using SAML 2.0.

> **Note:**
>
> The SAML 1.1 implementation does not use `HttpServletResponse` URL rewriting in SAML responses. Consequently, the `JSESSIONID` is not appended to SAML responses. However, means that SAML 1.1 cannot be used with browsers that do not support cookies.
>
> To enable `HttpServletResponse` URL rewriting, set the Java system property `weblogic.security.saml.enableURLRewriting` to `true`. For example, you can do this by specifying the following option in the Java command that starts WebLogic Server:
>
> `-Dweblogic.security.saml.enableURLRewriting=true`

Configuring a Source Site: Main Steps

To configure a WebLogic Server instance in the role of a source site, complete the following main steps:

1. Create and configure a SAML Credential Mapping provider V2 in your security realm.
2. Configure the federation services for the server instance in the realm that will serve as a source site.
3. Create and configure the relying parties for which SAML assertions will be produced.
4. If you want to require relying parties to use SSL certificates to connect to the source site, add any such certificates to the SAML credential mapping provider's Certificate Registry.

Configuring a Destination Site: Main Steps

To configure a WebLogic Server instance in the role of a destination site, complete the following main steps:

1. Create and configure a SAML Identity Assertion provider V2 in your security realm.
2. Configure the federation services for the server instance realm that will serve as a destination site.
3. Create and configure the asserting parties from which SAML assertions will be consumed.
4. Establish trust by registering the asserting parties' SSL certificates in the certificate registry maintained by the SAML Identity Assertion provider.

Configuring a SAML 1.1 Source Site for Single Sign-On

Learn how to configure a WebLogic Server instance as a SAML 1.1 source site.

> **Note:**
>
> The SAML 1.1 Credential Mapping provider and related configuration and services for SAML 1.1 federation services are deprecated as of WebLogic Server 14.1.2.0.0 and will be removed in a future release. Oracle recommends using SAML 2.0.

Configure the SAML 1.1 Credential Mapping Provider

In your security realm, create a SAML Credential Mapping Provider V2 instance. The SAML Credential Mapping provider is not part of the default security realm. See Configuring a SAML Credential Mapping Provider for SAML 1.1.

Configure the SAML Credential Mapping provider as a SAML authority, using the Issuer URI, Name Qualifier, and other attributes.

> **Note:**
>
> The SAML 1.1 Credential Mapping provider and related configuration and services for SAML 1.1 federation services are deprecated as of WebLogic Server 14.1.2.0.0 and will be removed in a future release. Oracle recommends using SAML 2.0.

Configure the Source Site Federation Services

Configuration of a WebLogic Server instance as a SAML 1.1 source site is controlled by the `FederationServicesMBean`. Access the `FederationServicesMBean` with the WebLogic Scripting Tool.

Configure SAML source site attributes as follows:

- **Enable the SAML Source Site**: Allow the WebLogic server instance to serve as a SAML source site by setting Source Site Enabled to true.
- **Set Source Site URL and Service URIs**: Set the URL for the SAML source site. This is the URL that hosts the Intersite Transfer Service and the Assertion Retrieval Service. The source site URL is encoded as a source ID in hex and Base64. When you configure a SAML Asserting Party for Browser/Artifact profile, you specify the encoded source ID.

  Specify the URIs for the Intersite Transfer Service and (to support Browser/Artifact profile) the Assertion Retrieval Service. (You also specify the Intersite Transfer Service URI when you configure a Relying Party.)

  The default URI `FederationServicesMBean.IntersiteTransferURIs` values are shown in Table 22-1.

Table 22-1 Intersite Transfer URIs

Default URI Values	Description
`/samlits_ba/its`	BASIC authentication, POST or Artifact profile
`/samlits_ba/its/post`	BASIC authentication, POST profile
`/samlits_ba/its/artifact`	BASIC authentication, Artifact profile
`/samlits_cc/its`	Client cert authentication, POST or Artifact profile

Table 22-1 (Cont.) Intersite Transfer URIs

Default URI Values	Description
`/samlits_cc/its/post`	Client cert authentication, POST profile
`/samlits_cc/its/artifact`	Client cert authentication, Artifact profile

The Intersite Transfer URI text box allows you to accept the default values as-is, or modify them as you choose. Each URI includes the application context, followed by `/its`, `/its/post`, or `/its/artifact`. The provided application contexts are `/samlits_ba` (BASIC authentication) or `/samlits_cc` (client certificate authentication). You could also specify an application-specific context if needed, for example `/yourapplication/its`, but in most cases the defaults provide the easiest configuration option.

If you specify these URIs as `/samlits_ba/its`, if a redirect occurs and the user's session on the source site has timed out, a BASIC authentication dialog is presented. If you instead want to use a FORM dialog, the URI should point to a custom Web application that authenticates users and then forwards to the actual ITS URI.

- **Add signing certificate**: The SAML source site requires a trusted certificate with which to sign assertions. Add this certificate to the keystore and enter the credentials (alias and passphrase) to be used to access the certificate. The server's SSL identity key/certificates will be used by default if a signing alias and passphrase are not supplied.
- **Configure SSL for the Assertion Retrieval Service**: You can require all access to the Assertion Retrieval Service to use SSL by setting `FederationServicesMBean.arsRequiresSSL` to `true`. You can require two-way SSL authentication for the Assertion Retrieval Service by setting both `arsRequiresSSL` and `ARSRequiresTwoWaySSL` to `true`.

Configure Relying Parties

A SAML Relying Party is an entity that relies on the information in a SAML assertion produced by the SAML source site. You can configure how WebLogic Server produces SAML assertions separately for each Relying Party or use the defaults established by the Federation Services source site configuration for producing assertion.

You can configure a Relying Party with the WebLogic Scripting Tool. See Configuring Relying and Asserting Parties with WLST.

The following topics explain how to configure Relying Parties:

- Configure Supported Profiles
- Assertion Consumer Parameters

Configure Supported Profiles

When you configure a SAML Relying Party, you can specify support for Artifact profile or POST profile, for the purposes of SAML SSO. As an alternative configure a Relying Party to support WSS/Holder-of-Key or WSS/Sender-Vouches profiles for Web Services Security purposes. Be sure to configure support for the profiles that the SAML destination sites support.

If you support the POST profile, optionally create a form to use in POST profile assertions for the Relying Party and set the pathname of that form in the POST Form attribute.

Assertion Consumer Parameters

For each SAML Relying Party, you can configure one or more optional query parameters that will be added to the ACS URL when redirecting to the destination site. In the case of POST profile, these parameters will be included as form variables when using the default POST form. If a custom POST form is in use, the parameters will be made available as a Map of names and values, but the form may or may not constructed to include the parameters in the POSTed data.

For WebLogic Server browser SSO configurations that communicate with another WebLogic Server instance, set the ID of the SAML Asserting Party (APID) in the relying party ACS parameters.

This parameter is required with the V2 providers in order for the browser profile configurations to work. That is, the ACS looks for an asserting party ID (APID) as a form parameter of the incoming request, and uses this to look up the configuration before performing any other processing.

The APID parameter also removes the need for you to specify a Target URL parameter for browser SSO. The Target URL is used for Web service configurations.

Replacing the Default Assertion Store

WebLogic Server uses a simple assertion store to maintain persistence for produced assertions. You can replace this assertion store with a custom assertion store class that implements `weblogic.security.providers.saml.AssertionStoreV2`. Configure WebLogic Server to use your custom assertion store class, rather than the default class, using the `FederationServicesMBean.AssertionStoreClassName` attribute. You can configure properties to be passed to the `initStore()` method of your custom assertion store class by using the `FederationServicesMBean.AssertionStoreProperties` attribute. Configure these attributes using the WebLogic Scripting Tool.

Configuring a SAML 1.1 Destination Site for Single Sign-On

Learn how to configure WebLogic Server as a SAML destination site.

> **Note:**
>
> The SAML 1.1 Identity Assertion provider and related configuration and services for SAML 1.1 federation services are deprecated as of WebLogic Server 14.1.2.0.0 and will be removed in a future release. Oracle recommends using SAML 2.0.

- Configure SAML Identity Assertion Provider
- Configure Destination Site Federation Services
- Configuring Asserting Parties

Configure SAML Identity Assertion Provider

In your security realm, create and configure a SAML Identity Assertion Provider V2 instance. The SAML Identity Assertion provider is not part of the default security realm. See Configuring a SAML Identity Assertion Provider for SAML 1.1.

> **Note:**
>
> The SAML 1.1 Identity Assertion provider (SAML Identity Assertion Provider V2) and related configuration and services for SAML 1.1 federation services are deprecated as of WebLogic Server 14.1.2.0.0 and will be removed in a future release. Oracle recommends using SAML 2.0.

Configure Destination Site Federation Services

Before you configure WebLogic as a SAML destination site, you must first create a SAML Identity Assertion Provider V2 instance in your security realm. Configuration of a WebLogic Server instance as a SAML destination site is controlled by the `FederationServicesMBean`. You can access the `FederationServicesMBean` using the WebLogic Scripting Tool.

Configure the SAML destination site attributes follows:

- Enable the SAML Destination Site
- Set Assertion Consumer URIs
- Specify Allowed Target Hosts
- Configure SSL for the Assertion Consumer Service
- Add SSL Client Identity Certificate
- Configure Single-Use Policy and the Used Assertion Cache or Custom Assertion Cache
- Configure Recipient Check for POST Profile

Enable the SAML Destination Site

Allow the WebLogic Server instance to serve as a SAML destination site by setting Destination Site Enabled to `true`.

Set Assertion Consumer URIs

Set the URIs for the SAML Assertion Consumer Service. This is the URL that receives assertions from source sites, so that the destination site can use the assertions to authenticate users. The Assertion Consumer URI is also specified in the configuration of a Relying Party.

Specify Allowed Target Hosts

Specify the list of allowed destination hosts where the target URL may be redirected by using the `FederationServicesMBean.AllowedTargetHosts` attribute. If no allowed target hosts are specified and the list is empty, then the target redirect URL will not be checked.

Configure SSL for the Assertion Consumer Service

You can require all access to the Assertion Consumer Service to use SSL by setting `FederationServicesMBean.acsRequiresSSL` to `true`.

Add SSL Client Identity Certificate

The SSL client identity is used to contact the ARS at the source site for Artifact profile. Add this certificate to the keystore and enter the credentials (alias and passphrase) to be used to access the certificate.

Configure Single-Use Policy and the Used Assertion Cache or Custom Assertion Cache

Optionally, you can require that each POST profile assertion be used no more than once. WebLogic Server maintains a cache of used assertions so that it can support a single-use policy for assertions. You can replace this assertion cache with a custom assertion cache class that implements `weblogic.security.providers.saml.SAMLUsedAssertionCache`. Configure WebLogic Server to use your custom assertion cache class, rather than the default class, using the `FederationServicesMBean.SAMLUsedAssertionCache` attribute. You can configure properties to be passed to the `initCache()` method of your custom assertion cache class using the `FederationServicesMBean.UsedAssertionCacheProperties` attribute. You can configure these attributes using the WebLogic Scripting Tool.

Configure Recipient Check for POST Profile

Optionally, you can require that the recipient of the SAML Response must match the URL in the HTTP Request. Do this by setting the POST Recipient Check Enabled attribute.

Configuring Asserting Parties

A SAML Asserting Party is a trusted SAML Authority (an entity that can authoritatively assert security information in the form of SAML Assertions).

You can configure an Asserting Party with the WebLogic Scripting Tool. See Configuring Relying and Asserting Parties with WLST.

The following topics explain key details about configuring an Asserting Party:

Configure Supported Profiles

When you configure a SAML Asserting Party, you can specify support for Artifact profile or POST profile, for the purposes of SAML SSO. Alternatively, configure an Asserting Party to support WSS/Holder-of-Key or WSS/Sender-Vouches profiles for Web Services Security purposes.

Configure Source Site ITS Parameters

For each SAML Asserting Party, configure zero or more optional query parameters that will be added when redirecting to the source site.

For WebLogic Server browser SSO configurations that communicate with another WebLogic Server instance, you must set the ID of the SAML Relying Party (RPID) in the Asserting Party ITS parameters.

This parameter is required with the V2 providers in order for the browser profile configurations to work. That is, the ITS looks for the RPID as a form parameter of the incoming request, and uses this to look up the configuration before performing any other processing.

The RPID parameter also removes the need for you to specify a Target URL parameter for WebLogic Server-to-WebLogic Server browser SSO configurations only. The Target URL is used for Web service configurations.

Configuring Relying and Asserting Parties with WLST

SAML partners (Relying Parties and Asserting Parties) are maintained in a registry. You can configure SAML partners using WebLogic Scripting Tool (WLST).

The following example shows how you might configure two Relying Parties using WLST in online mode. Note that the example sets the ID of the SAML Asserting Party (APID) in the relying party Assertion Consumer Service parameters. For WebLogic Server browser SSO configurations that communicate with another WebLogic Server instance, you must set the ID of the SAML Asserting Party (APID) in the relying party ACS parameters. (You would also set the ID of the SAML Relying Party (RPID) in the asserting party ITS parameters.)

The `demoidentity` certificate alias referenced in the example comes from the source site's demo SSL identity for the domain.

The APID is required for WebLogic Server-to-WebLogic Server browser SSO configurations only. This parameter is required with the V2 providers in order for the browser profile configurations to work.

Example 22-1 Creating Relying Parties with WLST

```
connect('','','t3://host:port')
Please enter your username :
Please enter your password :
...
rlm=cmo.getSecurityConfiguration().getDefaultRealm()
cm=rlm.lookupCredentialMapper('samlv2cm')

rp=cm.newRelyingParty()
rp.setDescription('test post profile')
rp.setProfile('Browser/POST')
rp.setAssertionConsumerURL('http://domain.example.com:7001/saml_destination/acs')
rp.setAssertionConsumerParams(array(['APID=ap_00001'],String))
rp.setSignedAssertions(true)
rp.setEnabled(true)
cm.addRelyingParty(rp)

rp=cm.newRelyingParty()
rp.setDescription('test artifact profile')
rp.setProfile('Browser/Artifact')
rp.setAssertionConsumerURL('http://domain.example.com:7001/saml_destination/acs')
rp.setAssertionConsumerParams(array(['APID=ap_00002'],String))
rp.setARSUsername('foo')
rp.setARSPassword('password')
rp.setSSLClientCertAlias('demoidentity')
rp.setEnabled(true)
cm.addRelyingParty(rp)
```

```
disconnect()
exit()
```

The following example shows how you might edit an existing Asserting Party. The example gets the Asserting Party, using its Asserting Party ID, and sets the Assertion Retrieval URL.

Example 22-2 Editing an Asserting Party with WLST

```
connect('','','t3://host:port')
Please enter your username :adminuser
Please enter your password :
...
rlm=cmo.getSecurityConfiguration().getDefaultRealm()
ia=rlm.lookupAuthenticationProvider('samlv2ia')
ap=ia.getAssertingParty('ap_00002')
ap.setAssertionRetrievalURL('https://hostname:7002/samlars/ars')
ia.updateAssertingParty(ap)
disconnect()
exit()
```

23

Configuring SAML 2.0 Services

Learn how to configure single sign-on in Oracle WebLogic Server with Web browsers and HTTP clients using SAML 2.0.

Configuring SAML 2.0 Services: Main Steps

Before you configure SAML 2.0 services, you must perform certain steps if you want to run this service in more than one WebLogic Server instance. You can then configure your WebLogic Server instance as either a Service Provider or Identity Provider.

A summary of the main steps you take to configure SAML 2.0 services is as follows:

1. Determine whether you plan to have SAML 2.0 services running in more than one WebLogic Server instance in the domain. If so, do the following:
   a. Create a domain in which the RDBMS security store is configured.

      The RDBMS security store is required by the SAML 2.0 security providers in production environments so that the data they manage can be synchronized across all the WebLogic Server instances that share that data.

      Note that Oracle does not recommend upgrading an existing domain in place to use the RDBMS security store. If you want to use the RDBMS security store, you should configure the RDBMS security store at the time of domain creation. If you have an existing domain with which you want to use the RDBMS security store, create the new domain and migrate your existing security realm to it.

      See Managing the RDBMS Security Store.

   b. Ensure that all SAML 2.0 services are configured identically in each WebLogic Server instance. If you are configuring SAML 2.0 services in a cluster, each Managed Server in that cluster must be configured individually.

   c. Note the considerations described in Web Application Deployment Considerations for SAML 2.0.
2. If you are configuring a SAML 2.0 Identity Provider site:
   a. Create and configure an instance of the SAML 2.0 Credential Mapping provider in the security realm.

   b. Configure the SAML 2.0 general services identically and individually in each WebLogic Server instance in the domain that will run SAML 2.0 services.

c. Configure the SAML 2.0 Identity Provider services identically and individually in each WebLogic Server instance in the domain that will run SAML 2.0 services.

d. Publish the metadata file describing your site, and manually distribute it to your Service Provider partners.

e. Create and configure your Service Provider partners.

3. If you are configuring a SAML 2.0 Service Provider site:

a. Create and configure an instance of the SAML 2.0 Identity Assertion provider in the security realm.

   If you are allowing virtual users to log in via SAML, you need to create and configure an instance of the SAML Authentication provider. See Configuring the SAML Authentication Provider.

b. Configure the SAML 2.0 general services identically and individually in each WebLogic Server instance in the domain that will run SAML 2.0 services.

c. Configure the SAML 2.0 Service Provider services identically and individually in each WebLogic Server instance in the domain that will run SAML 2.0 services.

d. Publish the metadata file describing your site, and manually distribute it to your Identity Provider partners.

e. Create and configure your Identity Provider partners.

The sections that follow provide details about each set of main steps.

Note:

- In this release of WebLogic Server, the SAML 2.0 implementation uses the SHA2 signature algorithm as the default for signing requests and responses. If required for backward compatibility, you can use the SHA1 signature algorithm by setting the Java system property `com.bea.common.security.saml2.useSHA1SigAlgorithm` to `true`. To do so, specify the following option in the Java command that starts WebLogic Server:

  ```
  -Dcom.bea.common.security.saml2.useSHA1SigAlgorithm=true
  ```

- In this release of WebLogic Server, the SAML 2.0 implementation no longer uses certificates that are expired or not yet valid in SAML signing. To allow use of these certificates, set the Java system property `com.bea.common.security.saml2.allowExpiredCerts` to `true`. For example, specify the following option in the Java command that starts WebLogic Server:

  ```
  -Dcom.bea.common.security.saml2.allowExpiredCerts=true
  ```

- The SAML 2.0 implementation does not use `HttpServletResponse` URL rewriting in SAML responses. Consequently, the `JSESSIONID` is not appended to SAML responses and, as a result, SAML 2.0 cannot be used with browsers that do not support cookies.

  To enable `HttpServletResponse` URL rewriting, set the Java system property `com.bea.common.security.saml2.enableURLRewriting` to `true`. For example, specify the following option in the Java command that starts WebLogic Server:

  ```
  -Dcom.bea.common.security.saml2.enableURLRewriting=true
  ```

Configuring SAML 2.0 General Services

Whether you configure a WebLogic Server instance as a SAML 2.0 Service Provider or as a SAML 2.0 Identity Provider, you must configure the server's general SAML 2.0 services using either the WebLogic Scripting Tool or WebLogic Remote Console. Configuration of the SAML 2.0 general services for a WebLogic Server instance is controlled by the `SingleSignOnServicesMBean`.

You can access the `SingleSignOnServicesMBean` with the WebLogic Scripting Tool or through WebLogic Remote Console (**SAML 2.0 General** attributes are located on the **Edit Tree** > **Environment** > **Servers** > *ServerName* page, under the **Security** tab.).

> **Note:**
>
> You cannot configure SAML 2.0 general services in a WebLogic Server instance until you have first configured either the SAML 2.0 Identity Assertion or SAML 2.0 Credential Mapping provider and restarted the server instance.

The following sections describe SAML 2.0 general services:

- About SAML 2.0 General Services
- Publishing and Distributing the Metadata File

About SAML 2.0 General Services

The general SAML 2.0 services you configure include the following:

- Whether you wish to enable the replicated cache

  Enabling the replicated cache is required if you are configuring SAML 2.0 services on two or more WebLogic Server instances in a domain, such as in a cluster. The replicated cache enables server instances to share and be synchronized with the data that is managed by the SAML 2.0 security providers; that is, either or both the SAML 2.0 Identity Assertion provider and the SAML 2.0 Credential Mapping provider.

  The RDBMS security store is required by the SAML 2.0 security providers in production environments so that the data they manage can be synchronized across all the WebLogic Server instances that share that data. (Use LDAP as the security store with the SAML 2.0 security providers only in development environments.)

  Therefore, prior to configuring SAML 2.0 services, the preferred approach is first to create a domain that is configured to use the RDBMS security store. See Managing the RDBMS Security Store.

- Information about the local site

  The site information you enter is primarily for the benefit of the business partners in the SAML federation with whom you share it. Site information includes details about the local contact person who is your partners' point of contact, your organization name, and your organization's URL.

- Published site URL

  This URL specifies the base URL that is used to construct endpoint URLs for the various SAML 2.0 services. The published site URL should specify the host name and port at

which the server is visible externally, which might not be the same at which the server is accessed locally. For example, if SAML 2.0 services are configured in a cluster, the host name and port may correspond to the load balancer or proxy server that distributes client requests to the Managed Servers in that cluster.

The published site URL should be appended with /saml2. For example:

```
https://www.avitek.com:7001/avitek-domain/aviserver/saml2
```

- Entity ID

  The entity ID is a human-readable string that uniquely distinguishes your site from the other partner sites in your federation. When your partners need to generate or consume an assertion, the SAML 2.0 services use the entity ID as part of the process of identifying the partner that corresponds with that assertion.

- Whether recipient check is enabled

  If enabled, the recipient of the authentication request or response must match the URL in the HTTP Request.

- Whether TLS/SSL client authentication is required for invocations on the Artifact Resolution Service. If enabled, SAML artifacts are encrypted when transmitted to partners.
- Transport Layer Security keystore alias and passphrase, the values used for securing outgoing communications with partners.
- Whether Basic authentication client authentication is required when your partners invoke the HTTPS bindings of the local site.

  If you enable this setting, you also specify the client username and password to be used. These credentials are then included in the published metadata file that you share with your federated partners.

- Whether requests for SAML artifacts received from your partners must be signed.
- Configuration settings for the SAML artifact cache.
- Keystore alias and passphrase for the key to be used when signing documents sent to your federated partners, such as authentication requests or responses.

For information about the steps for configuring SAML 2.0 general services, see Configure SAML 2.0 General Services in *Oracle WebLogic Remote Console Online Help*.

Publishing and Distributing the Metadata File

The local site information that is needed by your federated partners such as the local site contact information, entity ID, published site URL, whether TLS/SSL client authentication is required, and so on is published to a metadata file. See Publish SAML Metadata in *Oracle WebLogic Remote Console Online Help*.

When you publish the metadata file, you specify an existing directory on the local machine in which the file can be created. The process of distributing the metadata file to your federated partners is a detail that is not implemented by WebLogic Server. However, you may send this file via a number of commonly used mechanisms suitable for securely transferring electronic documents, such as encrypted email or secure FTP.

Keep the following in mind regarding the metadata file:

- Before you publish the metadata file, you should configure the Identity Provider and/or Service Provider services for the SAML 2.0 roles in which the WebLogic Server instances in your domain are enabled to function.

The configuration data for the SAML 2.0 services your site offers that is needed by your federated partners is included in this metadata file, greatly simplifying the tasks your partners perform to import your signing certificates, identify your site's SAML 2.0 service endpoints, including Single Logout, and use the correct binding types for connecting to your site's services, and so on.

- You should have only a single version of the metadata file that you share with your federated partners, even if your site functions in the role of Service Provider with some partners and Identity Provider with others. By having only a single version of the metadata file, you reduce the likelihood that your configuration settings might become incompatible with those of a partner.
- If you change the local site's SAML 2.0 configuration, you should update your metadata file. Because the metadata file is shared with your partners, it will be convenient to minimize the frequency with which you update your SAML 2.0 configuration so that your partners can minimize the need to make concomitant updates to their own partner registries.
- When you receive a metadata file from a federated partner, place it in a location that can be accessed by all the nodes in your domain in which SAML 2.0 services are configured. At the time you create a partner, you bring the contents the partner's metadata file into the partner registry.

Operations on the metadata file are available via the `com.bea.security.saml2.providers.registry.Partner` Java interface.

Configuring an Identity Provider Site for SAML 2.0 Single Sign-On

Before you configure SAML 2.0 Identity Provider services for your WebLogic Server instance, you must first configure a SAML 2.0 Credential Mapping provider instance in the security realm, and then configure SAML 2.0 general services. After performing these prerequisites, configure SAML 2.0 Identity Provider Services using the WebLogic Scripting Tool (WLST), or through WebLogic Remote Console.

> **Note:**
>
> When WebLogic Server is configured as an Identity Provider, it does not support SAML Single Logout.

This section presents the following topics:

- Configure the SAML 2.0 Credential Mapping Provider
- Configure SAML 2.0 Identity Provider Services
- Create and Configure Web Single Sign-On Service Provider Partners

Configure the SAML 2.0 Credential Mapping Provider

In your security realm, create a SAML 2.0 Credential Mapping provider instance. The SAML 2.0 Credential Mapping provider is not part of the default security realm. See Configuring a SAML 2.0 Credential Mapping Provider for SAML 2.0.

Configure the SAML 2.0 Credential Mapping provider as a SAML authority. Attributes you specify include the following:

- Issuer URI
- Name Qualifier
- Life span attributes for generated SAML 2.0 assertions
- Name mapper class name
- Whether generated assertions should include attribute information, which specify the groups to which the identity contained in the assertion belongs

After you configure the SAML 2.0 Credential Mapping provider, configure SAML 2.0 general services, as described in Configuring SAML 2.0 General Services.

Configure SAML 2.0 Identity Provider Services

Configuration of a WebLogic Server instance as a SAML 2.0 Identity Provider site is controlled by the `SingleSignOnServicesMBean`. You can access the `SingleSignOnServicesMBean` using the WebLogic Scripting Tool (WLST), or WebLogic Remote Console.

The sections that follow summarize the configuration tasks. For more information about performing these tasks, see Configure SAML 2.0 Identity Provider Services in *Oracle WebLogic Remote Console Online Help*.

Enable the SAML 2.0 Identity Provider Site

In WebLogic Remote Console, open the **Edit Tree** and go to **Environment**, then **Servers** and select the server you want to configure. On its **Security** tab, select the **SAML 2.0 Identity Provider** subtab. Turn on the **Enabled** option.

Specify if Authentication Requests Must Be Signed

Enable or disable the **Only Accept Signed Authentication Requests** attribute that determines whether incoming authentication requests must be signed. If enabled, then the authentication requests that are not signed are not accepted.

Specify a Custom Login Web Application

Optionally, you may use a custom login web application to authenticate users into the Identity Provider site. To configure a custom login web application, use the Login Customized attribute to specify the URL of the web application.

Enable Binding Types

Oracle recommends enabling all the available binding types for the endpoints of the Identity Provider services; namely, POST, Redirect, and Artifact. Optionally you may select a preferred binding type.

Configure Assertion Encryption

Set the following attributes to enable and configure encryption for SAML 2.0 assertions:

- Select **Assertion Encryption** to enable encryption for SAML 2.0 assertions.
- Optionally, update the default values of encryption algorithms in the **Key Encryption Algorithm** and the **Data Encryption Algorithm** fields.

Publish Your Site's Metadata File

After you have configured the SAML 2.0 general services and Identity Provider services, publish your site's metadata file and distribute it to your federated partners, as described in Publishing and Distributing the Metadata File.

Create and Configure Web Single Sign-On Service Provider Partners

A SAML 2.0 Service Provider partner is an entity that consumes the SAML 2.0 assertions generated by the Identity Provider site. You can configure Service Provider partners using WebLogic Remote Console.

The attributes that can be set on this console page can also be accessed programmatically via a set of Java interfaces, which are identified in the sections that follow.

See Create a SAML 2.0 Web Single Sign-On Service Provider Partner in *Oracle WebLogic Remote Console Online Help* for complete details about the specific steps for configuring a Service Provider partner. For a summary of the site information, signing certificates, and service endpoint information available when you configure a web single sign-on partner, see Viewing Partner Site, Certificate, and Service Endpoint Information.

This section includes the following topics:

Obtain Your Service Provider Partner's Metadata File

Before you configure a Service Provider partner for web single sign-on, you need to obtain the partner's SAML 2.0 metadata file via a trusted and secure mechanism, such as encrypted email or an SSL-enabled FTP site. Your partner's metadata file describes the partner site and binding support, includes the partner's certificates and keys, contains your partner's SAML 2.0 service endpoints, and more. Copy the partner's metadata file into a location that can be accessed by each node in your domain configured for SAML 2.0.

The SAML 2.0 metadata file is described in Publishing and Distributing the Metadata File.

Create Partner and Enable Interactions

To create and enable a Service Provider partner for web single sign-on, see Create a SAML 2.0 Web Single Sign-On Service Provider Partner in *Oracle WebLogic Remote Console Online Help*.

WebLogic Server provides the `com.bea.security.saml2.providers.registry.Partner` Java interface for configuring these attributes.

Configure How Assertions are Generated

Optionally, you can configure the following attributes of the SAML 2.0 assertions generated specifically for this Service Provider partner.

See Create a SAML 2.0 Web Single Sign-On Service Provider Partner in *Oracle WebLogic Remote Console Online Help*.

- The Service Provider Name Mapper Class name

  This is the Java class that overrides the default username mapper class with which the SAML 2.0 Credential Mapping provider is configured in this security realm.
- Time to Live attributes

The Time to Live attributes specify the interval of time during which the assertions generated for this partner are valid. These attributes prevent expired assertions from being used.

- Whether to generate attribute information that is included in assertions

  If enabled, the SAML 2.0 Credential Mapping provider adds, as attributes in the assertion, the groups to which the corresponding user belongs.
- Whether the assertions sent to this partner must be disposed of immediately after use
- Whether this server's signing certificate is included in assertions generated for this partner

WebLogic Server provides the `com.bea.security.saml2.providers.registry.SPPartner` Java interface for configuring these attributes.

Configure How Documents Are Signed

You can determine how the following documents exchanged with this partner must be signed.

See Create a SAML 2.0 Web Single Sign-On Service Provider Partner in *Oracle WebLogic Remote Console Online Help*.

- Assertions

  Operations on this attribute are available in the `com.bea.security.saml2.providers.registry.SPPartner` interface.
- Authentication requests

  Operations on this attribute are available in the `com.bea.security.saml2.providers.registry.WebSSOSPPartner` interface.
- Artifact requests

  Operations on this attribute are available in the `com.bea.security.saml2.providers.registry.WebSSOPartner` interface.

The attributes for specifying whether this partner accepts only signed assertions, or whether authentication requests must be signed, are read-only: they are derived from the partner's metadata file.

Configure Artifact Binding and Transport Settings

Optionally, you configure artifact binding and transport settings on the Service Provider partner page.

See Create a SAML 2.0 Web Single Sign-On Service Provider Partner in *Oracle WebLogic Remote Console Online Help*.

- Whether SAML artifacts are delivered to this partner via the HTTP POST binding. If so, you may also specify the URI of a custom web application that generates the HTTP POST form for sending the SAML artifact.
- The URI of a custom web application that generate the HTTP POST form for sending request or response messages via the POST binding.

Operations on these attributes are available via the `com.bea.security.saml2.providers.registry.WebSSOPartner` Java interface.

For added security in the exchange of documents with this partner, you can also specify a client user name and password to be used by the Service Provider partner when connecting to

the local site's binding using Basic authentication. This attribute is available via the `com.bea.security.saml2.providers.registry.BindingClientPartner` Java interface.

Configuring a Service Provider Site for SAML 2.0 Single Sign-On

As a prerequisite to configuring a SAML 2.0 Service Provider site, you must configure a SAML 2.0 Identity Assertion provider instance in your security realm, and then configure SAML 2.0 general services. If you plan to enable virtual users, you can optionally configure the SAML Authentication provider. After fulfilling the prerequisites, configure SAML 2.0 Service Provider Services using the WebLogic Scripting Tool (WLST) or WebLogic Remote Console.

> **Note:**
>
> As described in session-descriptor in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*, the cookie-path element defines the session tracking cookie path. If not set, this element defaults to / (slash), where the browser sends cookies to all URLs served by WebLogic Server.
>
> The WebLogic Server SAML 2.0 Service Providers require that the cookie-path be / (slash). If you set any other value for cookie-path, SSO fails for the SAML 2.0 Service Providers.

This section presents the following topics:

- Configure the SAML 2.0 Identity Assertion Provider
- Configure the SAML Authentication Provider
- Configure SAML 2.0 General Services
- Configure SAML 2.0 Service Provider Services
- Create and Configure Web Single Sign-On Identity Provider Partners

Configure the SAML 2.0 Identity Assertion Provider

In your security realm, create an instance of the SAML 2.0 Identity Assertion provider. The SAML 2.0 Identity Assertion provider is not part of the default security realm. The attributes you specify for the SAML 2.0 Identity Assertion provider include the following:

- Whether the replicated cache is enabled

  If you are configuring SAML 2.0 Identity Provider services in two or more server instances in the domain, this attribute must be enabled.
- A custom name mapper class that overrides the default SAML 2.0 assertion name mapper class

For more information about this security provider, see Configuring a SAML 2.0 Identity Assertion Provider for SAML 2.0.

Configure the SAML Authentication Provider

If you plan to enable virtual users, or consume attribute statements contained in assertions that you receive from your Identity Provider partners, you need to create and configure an instance of the SAML Authentication provider. See Configuring the SAML Authentication Provider.

Configure SAML 2.0 General Services

After configuring the SAML 2.0 Identity Assertion provider, and optionally the SAML Authentication provider, configure the SAML 2.0 general services, as described in Configuring SAML 2.0 General Services.

Configure SAML 2.0 Service Provider Services

Configuration of a WebLogic Server instance as a SAML 2.0 Service Provider site is controlled by the `SingleSignOnServicesMBean`. You can access the `SingleSignOnServicesMBean` using the WebLogic Scripting Tool (WLST) or WebLogic Remote Console.

You configure the SAML 2.0 Service Provider site attributes as summarized in the sections that follow. For more information about these configuration tasks, see Configure SAML 2.0 Service Provider Services in *Oracle WebLogic Remote Console Online Help*.

Enable the SAML 2.0 Service Provider Site

In WebLogic Remote Console, open the Edit Tree and go to **Environment**, then **Servers** and select the server you want to configure. On its **Security** tab, select the **SAML 2.0 Service Provider** subtab. Turn on the **Enabled** option.

Specify How Documents Must Be Signed

Optionally, you may enable or disable the following attributes that set the document signing requirements:

- **Always Sign Authentication Requests** that determines whether authentication requests sent to Identity Provider partners are signed.
- **Only Accept Signed Assertions** that determines whether assertions received from Identity Provider partners are signed. Note that this option is enabled by default to ensure that all incoming SAML 2.0 assertions must be signed.

Specify How Authentication Requests Are Managed

Optionally you may enable the following attributes of the authentication request cache:

- Maximum cache size
- Time-out value for authentication requests, which establishes the time interval beyond which stored authentication requests are expired

Enable Binding Types

Oracle recommends enabling all the available binding types for the endpoints of the Service Provider services; namely, POST, and Artifact. Optionally you may specify a preferred binding type.

Set Default URL

Optionally, you may specify the URL to which unsolicited authentication responses are sent if they do not contain an accompanying target URL.

Configure Assertion Encryption Key

Specify values for the following attributes to configure assertion encryption key:

- **Assertion Key Pass Phrase** that is required to retrieve the local site assertion key from the keystore
- **Assertion Key Alias** that is an alias for the keystore that contains the certificate and private key used to encrypt and decrypt the SAML assertions

Optionally, update the default list of encryption algorithms in the **Meta Data Encryption Algorithms** field.

Configure SAML Single Logout

SAML Single Logout (SLO) complements SAML Single Sign On by logging users out of all of the applications in their current SSO session at once. SAML SLO decreases the number of connections that remain active but unused, therefore reducing the opportunity for unauthorized access.

You can configure the behavior of the SAML SLO. Use the SLO Redirect URI option to determine where users are sent after they log out of an application. If no SLO Redirect URI is set, users are directed to the Default URL. You can also change the binding of the SAML SLO requests, which is set to HTTP Redirect by default, but can also be set to HTTP POST.

SAML SLO is enabled by default on WebLogic Server instances that act as Service Providers but you can choose to disable it if you want users to log out of each application separately.

Create and Configure Web Single Sign-On Identity Provider Partners

A SAML 2.0 Identity Provider partner is an entity that generates SAML 2.0 assertions consumed by the Service Provider site. You can configure Identity Provider partners using WebLogic Remote Console.

The attributes that can be set on this console page can also be accessed programmatically via a set of Java interfaces, which are identified in the sections that follow.

See Create a SAML 2.0 Web Single Sign-On Identity Provider Partner in *Oracle WebLogic Remote Console Online Help* for complete details about the specific steps for configuring a Service Provider partner.

For a summary of the site information, signing certificates, and service endpoint information available when you configure a web single sign-on partner, see Viewing Partner Site, Certificate, and Service Endpoint Information.

The following sections summarize tasks for configuring an Identity Provider partner:

Obtain Your Identity Provider Partner's Metadata File

Before you configure an Identity Provider partner for web single sign-on, you need to obtain the partner's SAML 2.0 metadata file via a trusted and secure mechanism, such as encrypted email or an SSL-enabled FTP site. Your partner's metadata file describes that partner site and binding support, includes the partner's certificates and keys, and so on. Copy the partner's metadata file into a location that can be accessed by each node in your domain configured for SAML 2.0.

The SAML 2.0 metadata file is described in Publishing and Distributing the Metadata File.

Create Partner and Enable Interactions

To create an Identity Provider partner and enable interactions for web single sign-on:

To create and enable an Identity Provider partner for web single sign-on, see Create a SAML 2.0 Web Single Sign-On Identity Provider Partner in *Oracle WebLogic Remote Console Online Help*.

WebLogic Server provides the `com.bea.security.saml2.providers.registry.Partner` Java interface for configuring these attributes.

Configure Authentication Requests and Assertions

Optionally, you can configure the following attributes of the authentication requests generated for, and assertions received from, this Identity Provider partner:

- The Identity Provider Name Mapper Class name

  This is the custom Java class that overrides the default username mapper class with which the SAML 2.0 Identity Assertion provider is configured in this security realm. The custom class you specify is used only for identities contained in assertions received from this particular partner.

  Operations on this attribute are available in the `com.bea.security.saml2.providers.registry.IdPPartner` Java interface.

- Whether the identities contained in assertions received from this partner are mapped to virtual users in the security realm

  > **Note:**
  >
  > To use this attribute, you must have a SAML Authentication provider configured in the realm.

  Operations on this attribute are available in the `com.bea.security.saml2.providers.registry.IdPPartner` Java interface.

- Whether to consume attribute information contained in assertions received from this partner

  If enabled, the SAML 2.0 Identity Assertion provider extracts attribute information from the assertion, which it uses in conjunction with the SAML Authentication provider (which must be configured in the security realm) to determine the groups in the security realm to which the corresponding user belongs.

  Operations on this attribute are available in the `com.bea.security.saml2.providers.registry.IdPPartner` Java interface.

- Whether authentication requests sent to this Identity Provider partner must be signed. This is a read-only attribute that is derived from the partner's metadata file.

  Operations on this attribute are available in the `com.bea.security.saml2.providers.registry.WebSSOIdPPartner` Java interface.

- Whether SAML artifact requests received from this Identity Provider partner must be signed.

Operations on this attribute are available in the `com.bea.security.saml2.providers.registry.WebSSOIdPPartner` Java interface.

Configure Redirect URIs

You can configure a set of URIs that, if invoked by an unauthenticated user, cause the user request to be redirected to the Identity Provider partner where the user can be authenticated.

> **Note:**
>
> If you configure one or more redirect URIs, remember to set a security policies on them as well; otherwise the web container will not attempt to authenticate the user and, consequently, not redirect the user's request to the Identity Provider partner.

WebLogic Server provides the `com.bea.security.saml2.providers.registry.WebSSOIdPPartner` Java interface for configuring this attribute.

Configure Binding and Transport Settings

Optionally, you configure artifact binding and transport settings on the Identity Provider partner page.

See Create a SAML 2.0 Web Single Sign-On Identity Provider Partner in *Oracle WebLogic Remote Console Online Help*.

- Whether SAML artifacts are delivered to this partner via the HTTP POST method. If so, you may also specify the URI of a custom web application that generates the HTTP POST form for sending the SAML artifact.
- The URL of the custom web application that generates the POST form for carrying the SAML response for POST bindings to this Identity Provider partner.
- The URL of the custom web application that generates the POST form for carrying the SAML response for Artifact bindings to this Identity Provider partner.

Operations on these attributes are available via the `com.bea.security.saml2.providers.registry.WebSSOPartner` Java interface.

For added security in the exchange of documents with this partner, you can also specify a client user name and password to be used by this Identity Provider partner when connecting to the local site's binding using Basic authentication. This attribute is available via the `com.bea.security.saml2.providers.registry.BindingClientPartner` Java interface.

Configuring SAML Encryption Using WLST

You can configure encryption for SAML 2.0 assertions using WLST scripts that perform operations on the `SingleSignOnServicesMBean`. Example 23-1 shows the use of WLST to enable encryption of SAML 2.0 assertions, and set the preferred key encryption and data encryption algorithms.

Example 23-1 Configure SAML Encryption Settings

```
edit()
startEdit()
```

```
srvr = cmo.lookupServer('myadmin')
realm = cmo.getSecurityConfiguration().getDefaultRealm()

###############################################################################
##
# SAML2 SSO Service Settings
###############################################################################
##
ssoSvc = srvr.getSingleSignOnServices()
ssoSvc.setAssertionEncryptionEnabled(true)
ssoSvc.setDataEncryptionAlgorithm('aes192-cbc')
ssoSvc.setKeyEncryptionAlgorithm('rsa-oaep')
```

Viewing Partner Site, Certificate, and Service Endpoint Information

When you configure SAML 2.0 partners, the partner configuration pages displayed by WebLogic Remote Console include sections for viewing and configuring additional information about the partner.

- The Site tab displays information about the Service Provider partner, which is derived from the partner's metadata file. The data in this tab is read-only.

  WebLogic Server provides the `com.bea.security.saml2.providers.registry.MetadataPartner` Java interface for partner site information.

- The Single Sign-On Signing Certificate tab displays details about the partner's signing certificate, which are also derived from the partner's metadata file. The data in this tab is read-only.

  Operations on these attributes are available from the `com.bea.security.saml2.providers.registry.WebSSOPartner` Java interface.

- The Transport Layer Client Certificate tab displays partner's transport layer client certificate. You can optionally import this certificate by clicking **Import Certificate from File**.

  Operations on this attribute are available from the `com.bea.security.saml2.providers.registry.BindingClientPartner` Java interface.

- When configuring Service Provider partners, the Assertion Consumer Service Endpoints tab is available, which displays the Service Provider partner's ACS endpoints. This data is also available from the `com.bea.security.saml2.providers.registry.WebSSOSPPartner` Java interface.

- When configuring Identity Provider partners, the Single Sign-On Service Endpoints tab is available, which displays the Identity Provider partner's single sign-on service endpoints. If the IdP partner supports SAML Single Logout, the Single Logout Endpoints tab is also available, which displays the Identity Provider partner's single logout service endpoints. This data is also available from the `com.bea.security.saml2.providers.registry.WebSSOIdPPartner` Java interface.

- The Artifact Resolution Service Endpoints tab displays the partner's ARS endpoints. This data is also available from the `com.bea.security.saml2.providers.registry.WebSSOPartner` Java interface.

Web Application Deployment Considerations for SAML 2.0

When deploying web applications for SAML-based SSO in a clustered environment, you must keep in mind certain considerations for preventing SAML-based single sign-on from failing.

Deployment Descriptor Recommendations

Note the following recommendations regarding the use of the following elements in deployment descriptor files:

- `relogin-enabled`
- `cookie-name`

This section includes the following topics:

Use of relogin-enabled with CLIENT-CERT Authentication

If a user logs in to a web application and tries to access a resource for which that user is not authorized, an HTTP FORBIDDEN (403) response is generated. This is standard web application behavior. However, for backwards compatibility with earlier releases, WebLogic Server permits web applications to use the `relogin-enabled` element in the `weblogic.xml` deployment descriptor file, so that the response to an access failure results in a request to authenticate. In certain circumstances, it can cause SAML 2.0 based web single sign-on to fail.

Normally, the SAML 2.0 Assertion Consumer Service (ACS) logs the user into the application and redirects the user request to the target web application. However, if that web application is enabled for SAML 2.0 single sign-on, is protected by `CLIENT-CERT` authentication, and has the `relogin-enabled` deployment descriptor element set to true, an infinite loop can occur in which a request to authenticate a user is issued repeatedly. This loop can occur when a user is logged in to the web application and attempts to access a resource for which the user is not permitted: instead of generating a FORBIDDEN message, a new authentication request is generated that triggers another SAML 2.0 based web single sign-on attempt.

To prevent this situation from occurring in a web application that is protected by `CLIENT-CERT` authentication, either remove the `relogin-enabled` deployment descriptor element for the web application, or set the element to `false`. This enables standard web application authentication behavior.

Use of Non-default Cookie Name

When the Assertion Consumer Service logs in the Subject contained in an assertion, an HTTP servlet session is created using the default cookie name `JSESSIONID`. After successfully processing the assertion, the ACS redirects the user's request to the target web application. If the target web application uses a cookie name other than `JSESSIONID`, the Subject's identity is not propagated to the target web application. As a result, the servlet container treats the user as if unauthenticated, and consequently issues an authentication request.

To avoid this situation, do not change the default cookie name when deploying web applications in a domain that are intended to be accessed by SAML 2.0 based single sign-on.

Login Application Considerations for Clustered Environments

Note the following two login limitations that are rare in clustered environments, but if they occur, they may prevent a single sign-on session from succeeding.

- When an Identity Provider's single sign-on service receives an authentication request, it redirects that request to the login application to authenticate the user. The login application must execute on the same cluster node as that single sign-on service. If not, the Identity Provider is unable to produce a SAML 2.0 assertion even if the authentication succeeds.

  Under normal circumstances, the login application executes on the same node as the single sign-on service, so likelihood of the authentication request being redirected to a login application executing on a different node in the domain is very small. However, it may happen if an authentication request is redirected by a cluster node different than the one hosting the login application. You can almost always prevent this situation from occurring if you configure the Identity Provider to use the default login URI with Basic authentication.

- When the SAML 2.0 Assertion Consumer Service (ACS) successfully consumes an assertion, it logs in the Subject represented by the assertion. The ACS then redirects the user request to the target application. Normally, the target application executes on the same node as the ACS. However, in rare circumstances, the target application to which is the user request is redirected executes on a cluster node other than the one hosting the ACS on which the login occurred. When this circumstance occurs, the identity represented by the assertion is not propagated to the target application node. The result is either another attempt at the single sign-on process, or denied access.

  Because the target application executes on the same node as the ACS, this situation is expected to occur very rarely.

Enabling Force Authentication and Passive Attributes is Invalid

When configuring SAML 2.0 Service Provider services, enabling both the Force Authentication and Passive attributes is an invalid configuration that WebLogic Server is unable to detect. If both these attributes are enabled, and an unauthenticated user attempts to access a resource that is hosted at the Service Provider site, an exception is generated and the single sign-on session fails.

Even if the user is already authenticated at the Identity Provider site and Force Authentication is enabled, the user is forced to authenticate again at the Identity Provider site.

Enabling SAML SLO on Web Applications

You can enable SAML SLO on a target web application deployed in a WebLogic Server instance acting as a Service Provider.

There are two ways to configure SAML SLO on a web application:

- Expose the SP SAML SLO init endpoint URL to authenticated SSO users. When authenticated SSO users access the URL, it triggers the SAML SLO process. You can also include the optional `slo_redirect_uri` parameter to specify where users are sent after logging out. For example, `/saml2/sp/slo/init[?slo_redirect_uri=<URI>]`.
- Configure the application's logout page to redirect to the SAML SLO init endpoint. Do not invoke the `logout()` method of the `javax.servlet.http.HttpServletRequest` object on the logout page to sign out users. The SLO init service will perform that step.

You can find the SAML SLO endpoint URL in the WebLogic Server published metadata file. For example:

```
<md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
Redirect" Location="https://<hostname>:<port>/saml2/sp/slo"/>
# or
<md:SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
POST" Location="https://<hostname>:<port>/saml2/sp/slo"/>
```

If the third-party SAML IdP does not support importing an SP partner's metadata, you must enter the WebLogic Server SLO endpoint URL manually and select one of the two supported bindings.

Enabling Synchronized Session Timeout

You can synchronize the expiration of all sessions that share the same session ID by enabling the `SynchronizedSessionTimeoutEnabled` attribute in either `WebAppContainerMBean` or `ServerTemplateMBean`.

For more information on configuring session timeout behavior, see Session Timeout in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

24

Enabling Debugging for SAML 1.1 and 2.0

Oracle Weblogic Server provides a variety of ways to enable debugging for a web application that uses SAML for SSO. Debugging is configured by setting attributes on the ServerDebug MBean.

This chapter includes the following topics:

About SAML Debug Scopes and Attributes

Learn about the registered debug scopes and attributes provided in WebLogic Server for SAML 1.1 and 2.0.

Table 24-1 SAML 1.1 Debug Scopes and Attributes

Scope	Attribute	Description
`weblogic.security.saml.atn`	`DebugSecuritySAMLAtn`	Prints information about SAML 1.1 authentication provider processing.
`weblogic.security.saml.credmap`	`DebugSecuritySAMLCredMap`	Prints information about SAML 1.1 credential mapping provider processing.
`weblogic.security.saml.lib`	`DebugSecuritySAMLLib`	Prints information about SAML 1.1 library processing.
`weblogic.security.saml.service`	`DebugSecuritySAMLService`	Prints information about SAML 1.1 SSO profile services.

Table 24-2 SAML 2.0 Debug Scopes and Attributes

Scope	Attribute	Description
`weblogic.security.saml2.atn`	`DebugSecuritySAML2Atn`	Prints information about SAML 2.0 authentication provider processing.
`weblogic.security.saml2.credmap`	`DebugSecuritySAML2CredMap`	Prints information about SAML 2.0 credential mapping provider processing.
`weblogic.security.saml2.lib`	`DebugSecuritySAML2Lib`	Prints information about SAML 2.0 library processing.
`weblogic.security.saml2.service`	`DebugSecuritySAML2Service`	Prints information about SAML 2.0 SSO profile services.

Enabling Debugging Using the Command Line

You can enable debug scopes or attributes by passing them as options in the command that starts WebLogic Server. This method for enabling SAML debugging is static and can only be used at server startup.

The command line options you can use for enabling SAML debugging by attribute are listed in Table 24-3.

Table 24-3 Command Line Options for SAML Debugging

SAML Version	Available Command Line Options for Debugging

SAML 1.1	`-Dweblogic.debug.DebugSecuritySAMLAtn=true`
	`-Dweblogic.debug.DebugSecuritySAMLCredMap=true`
	`-Dweblogic.debug.DebugSecuritySAMLLib=true`
	`-Dweblogic.debug.DebugSecuritySAMLService=true`

SAML 2.0	`-Dweblogic.debug.DebugSecuritySAML2Atn=true`
	`-Dweblogic.debug.DebugSecuritySAML2CredMap=true`
	`-Dweblogic.debug.DebugSecuritySAML2Lib=true`
	`-Dweblogic.debug.DebugSecuritySAML2Service=true`

Enabling Debugging Using WebLogic Remote Console

You can enable SAML debugging using WebLogic Remote Console. Using WebLogic Remote Console to enable or disable SAML debugging is dynamic and can be used while the server is running.

> **Note:**
>
> SAML 1.1 is deprecated as of WebLogic Server 14.1.2.0.0 and will be removed in a future release. Its configuration is not supported in WebLogic Remote Console. Oracle recommends using SAML 2.0 instead.

To configure SAML debugging using WebLogic Remote Console, complete the following steps:

1. In the **Edit Tree**, go to **Environment**, then **Servers**.
2. Click the server where you want to enable or disable debugging.
3. On the **Debug** tab, select the **Security** subtab.
4. Turn all of the SAML 2.0 debug attributes that you want to enable.

   For a description of each registered SAML debug attribute, see About SAML Debug Scopes and Attributes.
5. Click **Save**.

6. Repeat for the rest of the servers as desired.
7. Commit your changes.

Changes to SAML debug scopes and attributes take effect immediately, you do not need to restart the server. See Define Debug Settings in *Oracle WebLogic Remote Console Online Help*.

Enabling Debugging Using the WebLogic Scripting Tool

You can use the WebLogic Scripting Tool (WLST) to configure SAML debugging attributes. Using WLST is a dynamic method and can be used to enable debugging while the server is running.

For example, the following command runs a program for setting debugging attributes called `debug.py`:

```
java weblogic.WLST debug.py
```

The `debug.py` program contains the following code, which enables debugging for the attribute `DebugSecuritySAMLAtn`.

```
user='user1'
password='password'
url='t3://localhost:7001'
connect(user, password, url)
edit()
cd('Servers/myserver/ServerDebug/myserver')
startEdit()
set('DebugSecuritySAMLAtn','true')
save()
activate()
```

> **Note:**
>
> For clarity, this WLST example script shows the username and password in clear text. However, you should avoid entering clear-text passwords in WLST commands in general, and you should especially avoid saving on disk WLST scripts that include clear-text passwords. In these instances you should use a mechanism for passing encrypted passwords instead. See Security for WLST in *Understanding the WebLogic Scripting Tool*.

Note that you can also use WLST from Java. The following example shows the source file of a Java program that sets the `DebugSecuritySAMLAtn` debugging attribute:

```
import weblogic.management.scripting.utils.WLSTInterpreter;
import java.io.*;
import weblogic.jndi.Environment;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class test {
        public static void main(String args[]) {
       try {
              WLSTInterpreter interpreter = null;
              String user="user1";
              String pass="pw12ab";
```

```
                String url ="t3://localhost:7001";
                Environment env = new Environment();
                env.setProviderUrl(url);
                env.setSecurityPrincipal(user);
                env.setSecurityCredentials(pass);
                Context ctx = env.getInitialContext();

                interpreter = new WLSTInterpreter();
                interpreter.exec
                       ("connect('"+user+"','"+pass+"','"+url+"')");
                interpreter.exec("edit()");
                interpreter.exec("startEdit()");
                interpreter.exec
                       ("cd('Servers/myserver/ServerDebug/myserver')");
                interpreter.exec("set('DebugSecuritySAMLAtn','true')");
                interpreter.exec("save()");
                interpreter.exec("activate()");

        } catch (Exception e) {
        System.out.println("Exception "+e);
        }
        }
}
```

Sending Debug Messages to Standard Out

Messages corresponding to enabled debug attributes are sent to the server log file. Optionally, you can also send debug messages to standard out by passing the `StdoutSeverity=Debug` attribute on the LogMBean in the command to start WebLogic Server.
For example, `-Dweblogic.log.StdoutSeverity=Debug`. See Message Output and Logging in *Command Reference for Oracle WebLogic Server*.

Part V
Managing Security Information

Learn how to manage security information contained in the security store with which the Oracle WebLogic Server security realm is configured.

This part contains the following chapters:

25

Migrating Security Data

Learn how to export security data in Oracle WebLogic Server from one security realm or security provider and import the data into another realm or provider.

Overview of Security Data Migration

Security data (authentication, authorization, credential map, and role data) from one security realm can be exported into a file and then imported into another security realm. This data migration allows you to develop and test new security realms without recreating all the security data.

WebLogic security realms persist different kinds of security data — for example, users and groups (for the WebLogic Authentication provider), security policies (for the XACML Authorization provider), security roles (for the XACML Role Mapping provider), and credential maps (for the WebLogic Credential Mapping provider). When you configure a new security realm or a new security provider, you may prefer to use the security data from your existing realm or provider, rather than recreate all the users, groups, policies, roles, and credential maps. Several WebLogic security providers support security data migration. This means you can export security data from one security realm, and import it into a new security realm. You can migrate security data for each security provider individually, or migrate security data for all the WebLogic security providers at once (that is, security data for an entire security realm). Note that you can only migrate security data from one provider to another if the providers use the same data format. See Formats and Constraints Supported by WebLogic Security Providers. You migrate security data using the WebLogic Scripting Tool (WLST).

Migrating security data may be helpful when you:

- Transition from development to production mode.
- Copy production mode security configurations to security realms in new WebLogic domains.
- Move data from one security realm to a new security realm in the same WebLogic domain, where one or more of the default WebLogic security providers will be replaced with new security providers.

The remainder of this section describes security migration concepts, the formats and constraints supported by the WebLogic security providers, and steps for migrating security data with WLST.

Migration Concepts

Data migration concepts include format, constraints, and export files.

A *format* is a data format that specifies how security data should be exported or imported. *Supported formats* are the list of data formats that a given security provider understands how to process.

Constraints are key/value pairs that specify options to the export or import process. Use constraints to control which security data is exported to or imported from the security provider's database (in the case of the WebLogic Server security providers, the embedded LDAP server). For example, you may want to export only users (not groups) from an Authentication provider's database. *Supported constraints* are the list of constraints you can specify during the migration process for a particular security provider. For example, you can specify that an Authentication provider's database be used to import users and groups, but not security policies.

Export files are the files to which security data is written (in the specified format) during the export portion of the migration process. *Import files* are files from which security data is read (also in the specified format) during the import portion of the migration process. Both export and import files are simply temporary storage locations for security data as it is migrated from one security provider's data store to another security provider's data store.

Formats and Constraints Supported by WebLogic Security Providers

For security data to be exported and imported between security providers, both security providers must process the same format. Some data formats used for the WebLogic Server security providers are unpublished; therefore, you cannot currently migrate security data from a WebLogic security provider to a custom security provider, or vice versa, using the unpublished formats.

The following table identifies the import and export data formats that are supported by each of the WebLogic security providers

Table 25-1 Import and Export Formats Supported by the WebLogic Security Providers

WebLogic Provider	Supported Format
WebLogic Authentication provider	DefaultAtn—unpublished format
XACML Authorization Provider	XACML—standard XACML 2.0 format DefaultAtz—unpublished format
WebLogic Authorization Provider	DefaultAtz—unpublished format
XACML Role Mapping Provider	XACML—standard XACML 2.0 format DefaultRoles—unpublished format
WebLogic Role Mapping Provider	DefaultRoles—unpublished format
WebLogic Credential Mapping Provider	DefaultCreds—unpublished format
SAML Identity Asserter V2 SAML Credential Mapping Provider V2	XML Partner Registry—An XML format defined by the SAML partner registry schema JKS Key Store—A key store file format for importing and exporting partner certificates only LDIF Template—LDIF format

> **Note:**
>
> The WebLogic Authorization Provider and the WebLogic Role Mapping Provider are deprecated in WebLogic Server 14.1.1.0.0 and will be removed in a future release. Instead, the XACML Authorization provider and the XACML Role Mapping provider are the default providers.

WebLogic security providers support the import and export constraints provided in Table 25-2.

Table 25-2 Constraints Supported by the WebLogic Security Providers

WebLogic Security Provider	Supported Constraints	Description
Default Authentication	users groups	Export all users or all groups
• XACML Authorization • WebLogic Authorization • XACML Role Mapping • WebLogic Role Mapping	none	N/A
WebLogic Credential Mapping	passwords	With the constraint passwords=cleartext, passwords will be exported in clear text. Otherwise, they will be exported in encrypted form.
• SAML Identity Asserter V2 • SAML Credential Mapping V2	partners	Which partners to import or export. The constraint value can be one of: • all—all partners • none—no partners • list—only listed partners • enabled—only enabled partners • disabled—only disabled partners
• SAML Identity Asserter V2 • SAML Credential Mapping V2	certificates	Which certificates to import or export. The constraint value can be one of the following: • all—all certificates • none—no certificates • list—only listed certificates • referenced—only certificates referenced by a partner
• SAML Identity Asserter V2 • SAML Credential Mapping V2	passwords	With the constraint passwords=cleartext, passwords will be exported in clear text. Otherwise, they will be exported in encrypted form.

Table 25-2 (Cont.) Constraints Supported by the WebLogic Security Providers

WebLogic Security Provider	Supported Constraints	Description
• SAML Identity Asserter V2 • SAML Credential Mapping V2	importMode	Specifies how to resolve name conflicts between the imported data and existing data in the SAML registry. The constraint value can be one of the following: • fail—the import operation will fail if conflicts are detected (default) • rename—rename the imported entry that conflicts • replace—replace the existing entry with the conflicting imported entry

When exporting from the WebLogic Credential Mapping provider, SAML Credential Mapping provider, or SAML Identity Asserter, you need to specify whether or not the passwords for the credentials are exported in clear text. The constraint `passwords=cleartext` specifies that passwords will be exported in clear text. Otherwise, they will be exported in encrypted form. The mechanism used to encrypt passwords in each WebLogic domain is different; therefore, you want to export passwords in clear text if you plan to use them in a different WebLogic domain. After the credential maps are imported into the new WebLogic domain, the passwords are encrypted. Carefully protect the directory and file in which you export credential maps in clear text as secure data is available on your system during the migration process.

> **Note:**
>
> By default, the WebLogic Authentication provider stores passwords using a one-way hash. Passwords that have been encrypted by this provider cannot be unencrypted when you export data even if you use the `passwords=cleartext` constraint. If you want to be able to export passwords in clear text from this provider, you must set the `PasswordDigestEnabled` attribute on the `DefaultAuthenticatorMBean` to `true` prior to creating or updating those passwords.

Migrating Data with WLST

You can use the WebLogic Scripting Tool (WLST) to export and import data from a security provider. Access the Runtime MBean for the security provider and use its `importData` or `exportData` operation.

For example, you might use WLST to import data using commands like these:

```
serverConfig()
cd('SecurityConfiguration/mydomain/DefaultRealm/myrealm/path-to-MBean/mbeanname')
cmo.importData(format,filename,constraints)
```

where:

- *mbeanname*—Name of the security provider MBean.
- *format*—A format that is valid for the particular security provider. See Table 25-1.
- *filename*—The directory location and filename in which to export or import the security data. Remember that, regardless of whether you are using a UNIX or Windows operating

system, you need to use a forward slash, not a back slash, as a path separator for pathname arguments in WLST commands.

- *constraints*—The constraints that limit the data to be exported or imported

See *Understanding the WebLogic Scripting Tool*.

26

Managing the RDBMS Security Store

Oracle WebLogic Server provides an option of using an external RDBMS as a datastore for the authorization, role mapping, credential mapping, and certificate registry providers. This datastore, called the RDBMS security store, is strongly recommended for using SAML 2.0 services in two or more WebLogic Server instances in that domain, such as in a cluster. The RDBMS security store is required by the SAML 2.0 security providers in production environments so that the data they manage can be synchronized across all the WebLogic Server instances that share that data. (Use LDAP as the security store with the SAML 2.0 security providers only in development environments.)

> **Note:**
>
> In order to use the RDBMS security store, the preferred approach is first to create a domain in which the external RDBMS server is configured. Prior to booting the domain, you create the tables in the datastore that are required by the RDBMS security store. The WebLogic Server installation directory contains a set of SQL scripts that create these tables for each supported database.

This chapter presents the following topics:

For the most up-to-date details about the specific database systems that are supported for use as the RDBMS security store for WebLogic Server, see Oracle Fusion Middleware Supported System Configurations.

Security Providers that Use the RDBMS Security Store

Some WebLogic security providers use the RDBMS security store, if that store is configured in a domain.

The following is a list of such security providers:

- XACML Authorization provider
- XACML Role Mapping provider
- The following providers for SAML 1.1:
  - SAML Identity Assertion provider V2
  - SAML Credential Mapping provider V2
- The following providers for SAML 2.0:
  - SAML 2.0 Identity Assertion provider
  - SAML 2.0 Credential Mapping provider

- WebLogic Credential Mapping provider
- PKI Credential Mapping provider
- Certificate Registry

When the RDBMS security store is configured in a domain, an instance of any of the preceding security providers that has been created in the security realm automatically uses only the RDBMS security store as a datastore, and not the embedded LDAP server. WebLogic security providers configured in the domain that are not among those in the preceding list continue to use their respective default stores; for example, the Default Authentication provider continues to use the embedded LDAP server.

Oracle recommends that you configure the RDBMS security store at the time of domain creation, and before booting the domain. WebLogic Server includes the `RDBMSSecurityStoreMBean`, which is the interface for configuring the RDBMS security store via the WebLogic Scripting Tool (WLST). (The Configuration Wizard does not provide the ability to configure the RDBMS security store.)

Configuring the RDBMS Security Store

To create and configure the RDBMS security store, you must perform several tasks including, creating a domain with the RDBMS security store, creating RDBMS tables in the security datastore, configuring a JMS topic for the RDBMS security store and, so on.

The following topics describe the tasks you need to perform in order to configure the RDBMS security store:

- Create a Domain with the RDBMS Security Store
- Create RDBMS Tables in the Security Datastore
- Configure a JMS Topic for the RDBMS Security Store

Create a Domain with the RDBMS Security Store

To use the RDBMS security store in a domain, Oracle recommends that you configure the RDBMS security store at the time you create that domain. Oracle does not recommend modifying an existing domain in place to use the RDBMS as the security store. If the database connection is not configured correctly, the policies necessary for granting access to the domain could become unavailable, resulting in a domain that cannot be used.

The high-level process for creating a domain to use an RDBMS security store is as follows:

1. Create a new domain to use an RDBMS security store in either of the following ways:
   - Use WLST offline to create the domain and configure the RDBMS security store using a single script.
   - Create a new WebLogic domain using the Configuration Wizard or WLST. Then, before starting the domain, use WLST offline to configure the RDBMS security store. The Configuration Wizard does not provide the ability to configure the RDBMS security store. See Creating and Configuring the WebLogic Domain in *Installing and Configuring Oracle WebLogic Server and Coherence*.

   Sample scripts for configuring the RDBMS security store using WLST offline are provided in Use WLST Offline to Create the RDBMS Security Store.
2. Prior to starting the domain, create the RDBMS tables in your datastore. The WebLogic Server installation directory includes a set of scripts for each supported RDBMS system.

Typically this step is performed by the database administrator. See Create RDBMS Tables in the Security Datastore.

3. Test the database connection using the Java utility `dbping`. See Testing the Database Connection.
4. Start the domain.

> **Note:**
>
> The domain will not start if the RDBMS tables are not created or the database connection is not available.

5. After you start the domain, you can manage the RDBMS configuration using WebLogic Remote Console. See Configure the RDBMS Security Store in *Oracle WebLogic Remote Console Online Help*.
6. If the RDBMS security store is configured in a domain that includes two or more WebLogic Server instances, or a cluster, Oracle strongly recommends that you enable JMS notifications for that domain and configure a JMS topic that can be used by the RDBMS security store. See Configure a JMS Topic for the RDBMS Security Store.
7. If you are configuring single sign-on using SAML, see Configuring SAML 2.0 Services. The RDBMS security store is required by the SAML 2.0 security providers in production environments so that the data they manage can be synchronized across all the WebLogic Server instances that share that data.

Use WLST Offline to Create the RDBMS Security Store

You can use WLST offline to create the RDBMS security store in either of the following ways:

- Use a single WLST offline script to create the domain and configure the RDBMS security store.
- Create the domain using the Configuration Wizard, then use WLST offline to configure the RDBMS security store.

To configure the RDBMS security store, you need to specify the following database connection properties in the WLST scripts:

- RDBMS type

  For information about the databases that are supported for containing the RDBMS security store, see Oracle Fusion Middleware Supported System Configurations.
- JDBC driver and class name for connecting to the RDBMS, for example `oracle.jdbc.OracleDriver`
- RDBMS name, host, port, and URL
- User name and password of the domain user who can access the RDBMS system

> **Note:**
>
> For clarity, the WLST examples provided in this section show passing the user name and password credentials of the RDBMS system user in clear text. However, you should avoid entering clear-text passwords in WLST commands in general, and you should especially avoid saving on disk WLST scripts that include clear-text passwords. In these instances you should use a mechanism for passing encrypted passwords instead.

- Optionally, any connection properties that need to be passed to the RDBMS system. The parameters that you specify in the JDBC driver connection properties attribute must be a comma-separated list.

> **Note:**
>
> Operations for creating and configuring the RDBMS security store are available using the `RDBMSSecurityStoreMBean`. Internally, the RDBMS security store connects to and interoperates with the database using the JDBC driver supported by the database. The attributes set on the `RDBMSSecurityStoreMBean` are converted into attributes set on the `javax.sql.DataSource` implementation. For more information about the attributes that you can set on the `RDBMSSecurityStoreMBean`, see RDBMSSecurityStoreMBean in the *MBean Reference for Oracle WebLogic Server*.

The following WLST offline scripts provide examples that demonstrate how to create a domain and configure the RDBMS security store in a single script, as well as how to configure the RDBMS security store for Oracle, MS-SQL, and DB2 after you have created a new domain using the Configuration Wizard or WLST:

- Create Domain with RDBMS Security Store Example provides an example of a WLST offline script that combines domain creation and RDBMS security store configuration into a single script. This script configures an MS/SQL database as the RDBMS security store.
- Oracle Database Example includes an example WLST offline script for configuring an Oracle Database as the RDBMS security store. Execute this script after creating a new domain using the Configuration Wizard or WLST, and before starting the domain.
- MS-SQL Example includes an example WLST offline script for configuring a MS-SQL database as the RDBMS security store. Execute this script after creating a new domain using the Configuration Wizard or WLST, and before starting the domain.
- DB2 Example includes an example WLST offline script for configuring DB2 as the RDBMS security store. Execute this script after creating a new domain using the Configuration Wizard or WLST, and before starting the domain.

You should save your WLST scripts using a `.py` file extension, for example *`filename`*`.py`. To run these scripts, include the name of the script when starting WLST offline. For example, to start WLST offline on Linux:

```
cd ORACLE_HOME/oracle_common/common/bin
./wlst.sh filename.py
```

In this example, *`ORACLE_HOME`* represents the Oracle Home directory you specified during installation.

See Running Scripts in *Understanding the WebLogic Scripting Tool*.

Create Domain with RDBMS Security Store Example

Example 26-1 provides an example WLST script to create a domain named base_domain, and to configure an MS/SQL database as the RDBMS security store.

Example 26-1 Creating a Domain and Configuring MS-SQL as the RDBMS Security Store

```
# Select and load the template to use for creating the domain
selectTemplate('Basic WebLogic Server Domain')
loadTemplates()
cd('/')

# Set the name of the domain as base_domain
set('Name', 'base_domain')

# Set the listen port for the Administration Server
cd('Servers/AdminServer')
set('Name', 'admin')
set('ListenPort',7001)

# Set the user name and password for the WebLogic Server administration user
cd('/')
cd('Security')
cd('base_domain')
cd('User/adminusername')
cmo.setName('adminusername')
cmo.setPassword('adminpassword')

# Create the RDBMS security store
print("configure RDBMS store")
create('base_domain','SecurityConfiguration')
cd('/SecurityConfiguration/base_domain')
cd('Realm/myrealm')

# Specify the database attributes on the RDBMSSecurityStoreMBean
rdbms.setUsername('DBuser')
rdbms.setPasswordEncrypted('DBpassword')
rdbms.setConnectionURL('jdbc:weblogic:sqlserver://host.example.com:port)
rdbms.setDriverName('weblogic.jdbc.sqlserver.SQLServerDriver')
rdbms.setConnectionProperties('user=DBuser,portNumber=port,databaseName=DbName,serverName
=host.example.com')

# Write the domain
writeDomain('/home/domains/base_domain')

# Exit WLST offline
exit()
```

Oracle Database Example

Example 26-2 shows an example WLST script for configuring an Oracle Database as the RDBMS security store. Execute this script after creating a new domain using the Configuration Wizard or WLST, and before starting the domain.

Example 26-2 Configuring Oracle Database for the RDBMS Security Store

```
# Read the domain using the complete path to the domain directory
readDomain(domainDirName)
```

```
# Create the RDBMS security store
create('DomainName','SecurityConfiguration')
cd('/SecurityConfiguration/DomainName')
cd('Realm/myrealm')
rdbms = create('myRDBMSSecurityStore','RDBMSSecurityStore')

# Specify the database attributes on the RDBMSSecurityStoreMBean
rdbms.setUsername('DBuser')
rdbms.setPasswordEncrypted('DBpassword')
rdbms.setConnectionURL('jdbc:oracle:thin:@hostname.domain:port:sid')
rdbms.setDriverName('oracle.jdbc.OracleDriver')
rdbms.setConnectionProperties('user=DBuser,portNumber=port,SID=sid,serverName=hostname.do
main')

# Update the domain
updateDomain()

# Exit WLST offline
exit()
```

MS-SQL Example

Example 26-3 shows an example WLST script for configuring an MS-SQL database as the RDBMS security store. Execute this script after creating a new domain using the Configuration Wizard or WLST, and before starting the domain.

Example 26-3 Configuring MS-SQL for the RDBMS Security Store

```
# Read the domain using the complete path to the domain directory
readDomain(DomainDirName)

# Create the RDBMS security store
create('DomainName','SecurityConfiguration')
cd('/SecurityConfiguration/DomainName')
cd('Realm/myrealm')
rdbms = create('myRDBMSSecurityStore','RDBMSSecurityStore')

# Specify the database attributes on the RDBMSSecurityStoreMBean
rdbms.setUsername('DBuser')
rdbms.setPasswordEncrypted('DBpassword')
rdbms.setConnectionURL('jdbc:weblogic:sqlserver://ServerName:port')
rdbms.setDriverName('weblogic.jdbc.sqlserver.SQLServerDriver')
rdbms.setConnectionProperties('user=DBuser,portNumber=port,databaseName=DBname,serverName
=ServerName')

# Update the domain
updateDomain()

# Exit WLST offline
exit()
```

DB2 Example

Example 26-4 shows an example WLST script for configuring DB2 as the RDBMS security store. Execute this script after creating a new domain using the Configuration Wizard or WLST, and before starting the domain.

> **Note:**
>
> If you choose DB2, you have the option of selecting the WebLogic Type 4 JDBC driver for DB2 that is provided in WebLogic Server. However, if you use this JDBC driver, you must also specify the additional property `BatchPerformanceWorkaround` and set it to `true`. If you do not set the `BatchPerformanceWorkaround` to `true` in this configuration, WebLogic Server may fail to boot, generating a SecurityServiceException message.

Example 26-4 Configuring DB2 for the RDBMS Security Store

```
# Read the domain using the complete path to the domain directory
readDomain(domainDirName)

Create the RDBMS security store
create('DomainName','SecurityConfiguration')
cd('/SecurityConfiguration/DomainName')
cd('Realm/myrealm')
rdbms = create('myRDBMSSecurityStore','RDBMSSecurityStore')

# Specify the database attributes on the RDBMSSecurityStoreMBean
rdbms.setUsername('DBuser')
rdbms.setPasswordEncrypted('DBpassword')
rdbms.setConnectionURL('jdbc:weblogic:db2://ServerName:port')
rdbms.setDriverName('weblogic.jdbc.db2.DB2Driver')
rdbms.setConnectionProperties('user=DBuser,portNumber=port,databaseName=DBname,AlternateI
D=DBname-alt,serverName=ServerName,batchPerformanceWorkaround=true')

# Update the domain
updateDomain()

# Exit WLST offline
exit()
```

For more information about specifying connection properties for the WebLogic Type 4 JDBC driver for DB2, see Using DataDirect Documentation in *Developing JDBC Applications for Oracle WebLogic Server*.

Testing the Database Connection

When you configure an RDBMS security store, Oracle strongly recommends testing the database connection to verify that the connection is set up properly. If there were a problem with the database connection, you might not be able to boot the domain if the security providers that control access to that domain are unable to obtain the necessary security policies.

You can test the database connection using the WebLogic Server Java utility `dbping`. The `dbping` command-line utility tests the connection between the RDBMS and your client machine using a JDBC driver. See dbping in *Command Reference for Oracle WebLogic Server*.

> **Note:**
>
> Before running the `dbping` utility, be sure to include the JDK in your `PATH` environment variable, then run the `setDomainEnv` script to set up your environment.

The following example shows how to execute the `dbping` utility using an Oracle Thin Driver connected to an Oracle Database. The example includes steps for setting up your environment on a Linux host before executing the `dbping` utility.

```
$ bash
$ export PATH=$JAVA_HOME/bin:$PATH
$ cd myDomain/bin
$ . ./setDomainEnv.sh
$ java utils.dbping ORACLE_THIN DBuser DBpassword
myhost.example.com:port:DemoDB

**** Success!!! ****

You can connect to the database in your app using:

  java.util.Properties props = new java.util.Properties();
  props.put("user", "DBuser");
  props.put("password", "DBpassword");
  java.sql.Driver d =
    Class.forName("oracle.jdbc.OracleDriver").newInstance();
  java.sql.Connection conn =
    Driver.connect("DBuser", props);
```

Create RDBMS Tables in the Security Datastore

Prior to booting the domain, the database administrator needs to run the SQL script that creates the RDBMS tables in the datastore used by the RDBMS security store. A set of SQL scripts for creating these tables for each supported RDBMS system is available in the following WebLogic Server installation directory. SQL scripts for removing the tables are also provided.

WL_HOME`/server/lib`

When running the appropriate SQL script for the database serving as the RDBMS security store, be sure to specify the same connection properties, including the credentials of the user who has access, the database URL, and so on, as specified for that RDBMS during domain creation.

Table 26-1 identifies the name of each of these SQL scripts. For specific database versions supported, see Oracle Fusion Middleware Supported System Configurations.

Table 26-1 SQL Scripts for Creating and Removing RDBMS Datastore Tables

RDBMS	Script for Creating Datastore Tables	Script for Removing Datastore Tables
Oracle	`rdbms_security_store_oracle.sql`	`rdbms_security_store_oracle_remove.sql`

Table 26-1 (Cont.) SQL Scripts for Creating and Removing RDBMS Datastore Tables

RDBMS	Script for Creating Datastore Tables	Script for Removing Datastore Tables
MS-SQL	`rdbms_security_store_sqlserver.sql`	`rdbms_security_store_sqlserver_remove.sql`
DB2	`rdbms_security_store_db2.sql`	`rdbms_security_store_db2_remove.sql`
Derby	`rdbms_security_store_derby.sql`	`rdbms_security_store_derby_remove.sql`

Configure a JMS Topic for the RDBMS Security Store

If the RDBMS security store is configured in a domain that includes two or more WebLogic Server instances, or a cluster, Oracle strongly recommends that you also perform the following tasks:

1. Enable JMS notifications for that domain.
2. Configure a JMS topic that can be used by the RDBMS security store.

JMS notifications enable the security data that is contained in the RDBMS security store, and that is managed by security providers in the realm, to be synchronized among all server instances in the domain.

> **Note:**
>
> If you do not configure a JMS topic that can be used by the RDBMS security store when configured in a multi-server or clustered domain, care should be taken when making security policy or security configuration updates. If no JMS topic is configured, it may be necessary to reboot the domain to ensure that all server instances function consistently with regards to those security updates.

You can enable JMS notifications by booting the domain in which the RDBMS security store has been configured, and configuring attributes on the `RDBMSSecurityStoreMBean` using either of the following mechanisms:

- WebLogic Scripting Tool
- In WebLogic Remote Console, open the **Edit Tree** and go to **Security**, then **Realms**, then *myRealm*, then **RDBMS Security Store**.

The attributes of the `RDBMSSecurityStoreMBean` that must be set to enable JMS notifications are listed and described in Table 26-2.

Table 26-2 RDBMSSecurityStoreMBean Attributes for Configuring a JMS Topic

Attribute Name	Description
`JMSTopic`	The JMS topic to which notifications are published to and to which notifications sent from other JVMs are subscribed. The target JMS topic needs to be pre-deployed.

Table 26-2 (Cont.) RDBMSSecurityStoreMBean Attributes for Configuring a JMS Topic

Attribute Name	Description
JMSTopicConnectionFactory	The JNDI name of a javax.jms.TopicConnectionFactory instance to use for finding JMS topics. The topic Connection Factory Configuration in *Administering JMS Resources for Oracle WebLogic Server* describes the WebLogic JMS connection factory, weblogic.jms.ConnectionFactory, which is a javax.jms.TopicConnectionFactory instance. Refer to this topic for information about configuring a connection factory.
NotificationProperties	A comma-delimited list of key-value properties to pass to the JNDI InitialContext on construction, in the form of *xx*Key=*xx*Value, *xx*Key=*xx*Value. The following properties must be specified: • java.naming.provider.url — Property for specifying configuration information for the service provider to use. The value of the property should contain a URL string. For example: iiops://*localhost*:7002 • java.naming.factory.initial — Property for specifying the initial context factory to use. The value of the property should be the fully-qualified class name of the factory class that will create an initial context. For example: weblogic.jndi.WLInitialContextFactory
JNDIUserName	The identity of any valid user in the security realm who has access to JNDI.
JNDIPassword	The password of the user specified in the JNDIUserName attribute.
JMSExceptionReconnectAttempts	The number of reconnect attempts to be made if the JMS system detects a serious connection error. The default is 0, which causes an error to be logged, but does not result in a reconnect attempt.

See the following topics:

- Configure Resources for JMS System Modules in *Oracle WebLogic Remote Console Online Help*
- Configuring Basic JMS System Resources in *Administering JMS Resources for Oracle WebLogic Server*
- Configure the RDBMS Security Store in *Oracle WebLogic Remote Console Online Help*
- RDBMSSecurityStoreMBean in the *MBean Reference for Oracle WebLogic Server*

Configuring JMS Connection Recovery in the Event of Failure

Normally, the WebLogic Security Service contained in each WebLogic Server instance in a multi-node domain connects at startup to the JMS server. If a security provider that uses the RDBMS security store makes a change to its security data, all WebLogic Server instances are notified via JMS, and the local caches used by the WebLogic Security Service in each server instance are synchronized to that change.

If the JMS connection fails in a WebLogic Server instance that has been successfully started, the WebLogic Security Service associated with that server instance starts the JMS connection recovery process. The recovery process sleeps one second between reconnect attempts. The recovery process is stopped if the JMS connection failure persists after the number of reconnect attempts with which the JMSExceptionReconnectAttempts property has been configured is reached. No further reconnect attempts are made: If a change is made to the

security data in one WebLogic Server instance, the local caches managed by the WebLogic Security Service in other WebLogic Server instances are not synchronized to that change. However, if the JMS connection is successfully recovered by other means (such as a server reboot), those caches become synchronized.

If the JMS connection is not successfully started at the time a WebLogic Server instance is booted, a timer task that makes reconnect attempts is automatically started. The timer task is cancelled once the connection is successfully made. Two system properties may be configured for this timer task:

- `com.bea.common.security.jms.initialConnectionRecoverInterval`

  Specifies the delay, in milliseconds, before the connection recovery task is executed. The default value is 1000, which causes the connection recovery process to be executed after a delay of one second.

- `com.bea.common.security.jms.initialConnectionRecoverAttempts`

  Specifies the maximum number of reconnect attempts that can be made prior to cancelling the timer task. The default value is 3600, which causes the timer task to be cancelled once 3600 reconnect attempts have been made. No further reconnect attempts are made.

You can calculate the maximum connection polling duration by multiplying the values specified by each of the preceding system properties. For example, multiplying the default values of these two properties yields a maximum polling duration of one hour (1000 millisecond delay multiplied by 3600 reconnect attempts).

Upgrading a Domain to Use the RDBMS Security Store

To upgrade a domain to use the RDBMS security store, Oracle recommends creating a new domain in which the RDBMS security store is configured. After you create the new domain, you should export the security data from the security realm of the old domain, and import it into a security realm of the new domain.

> **Note:**
>
> For details about creating a new domain and configuring the RDBMS security store, see Create a Domain with the RDBMS Security Store.

When you import security data into a security realm in a domain that uses the RDBMS security store, the data for the security providers that use the RDBMS security store is automatically loaded into that datastore. Data for security providers that do not use the RDBMS security store is automatically imported into the stores that those providers normally use by default.

It is possible to selectively migrate security providers individually from one security realm to another. However, when migrating security data to a domain that uses the RDBMS security store, Oracle recommends migrating the security realm's data in a single operation.

For information about migrating security realms, see Migrating Security Data.

27

Managing the Embedded LDAP Server

Learn how to configure and manage the embedded LDAP server which is the data store for the default Authentication, Authorization, Credential Mapping, and Role Mapping providers included in Oracle WebLogic Server.

Configuring the Embedded LDAP Server

The embedded LDAP server contains user, group, group membership, security role, security policy, and credential map information. By default, each WebLogic domain has an embedded LDAP server configured with the default values set for each type of information.

The Default Authentication, Authorization, Credential Mapping, and Role Mapping providers use the embedded LDAP server as their data store. If you use any of these providers in a new security realm, you may want to change the default values for the embedded LDAP server to optimize its use in your environment.

> **Note:**
>
> The performance of the embedded LDAP server is best with fewer than 10,000 users. If you have more users, consider using a different LDAP server and Authentication provider.

See Configure the Embedded LDAP Server in *Oracle WebLogic Remote Console Online Help*.

The data file and change log file used by the embedded LDAP server can potentially grow quite large. You can configure maximum sizes for these files with the following `weblogic.Server` command line arguments:

- `-Dweblogic.security.ldap.maxSize=<max bytes>`, which limits the size of the data file used by the embedded LDAP server. When the data file exceeds the specified size, WebLogic Server eliminates from the data file space occupied by deleted entries.
- `-Dweblogic.security.ldap.changeLogThreshold=<number of entries>`, which limits the size of the change log file used by the embedded LDAP server. When the change log file exceeds the specified number of entries, WebLogic Server truncates the change log by removing all entries that have been sent to all Managed Servers.

Embedded LDAP Server Replication

The embedded LDAP server for a domain consists of a primary LDAP server, maintained in the domain's Administration Server, and a replicated LDAP server maintained in each Managed Server in the domain. You can manage the behavior of the embedded LDAP server on the Administration Server and Managed Servers using WebLogic Remote Console.

When changes are made using a Managed Server, updates are sent to the embedded LDAP server on the Administration Server. The embedded LDAP server on the Administration Server maintains a log of all changes. The embedded LDAP server on the Administration Server also maintains a list of Managed Servers and the current change status for each one. The embedded LDAP server on the Administration Server sends appropriate changes to each Managed Server and updates the change status for each server. This process occurs when an update is made to the embedded LDAP server on the Administration Server. However, depending on the number of updates, it may take several seconds or more for the change to be replicated to the Managed Server.

You can configure the behavior of the embedded LDAP server on the Administration Server and the Managed Servers in a domain. In the WebLogic Remote Console **Edit Tree**, go to go to **Environment**, then **Domain**. Then select the **Security** tab and the **Embedded LDAP** subtab. You can set these attributes:

- **Refresh Replica At Startup** — Specifies whether the embedded LDAP server in a Managed Server should refresh all replicated data at boot time. This setting is useful if you have made many changes when the Managed Server was not active, and you want to download the entire replica instead of having the Administration Server push each change to the Managed Server.
- **Master First** — Specifies whether a Managed Server should always connect to the primary embedded LDAP server on the Administration Server, instead of connecting to the local replicated LDAP server.

See Configure the Embedded LDAP Server in *Oracle WebLogic Remote Console Online Help*.

> **Note:**
>
> Deleting and modifying the configured security providers through WebLogic Remote Console may require manual clean up of the embedded LDAP server. Use an external LDAP browser to delete unnecessary information.

Viewing the Contents of the Embedded LDAP Server from an LDAP Browser

To view the contents of the embedded LDAP server through an LDAP browser, you must have access to an external LDAP browser, change the credential for the embedded LDAP server, and configure a new connection in the LDAP browser.

The steps for viewing the contents of the embedded LDAP server through an LDAP browser are described here:

1. If you don't already have one, you can download and install any external LDAP browser of your choice.

2. In WebLogic Remote Console, change the credential for the embedded LDAP server:
   a. In the **Edit Tree**, go to **Environment**, then **Domain**.
   b. On the **Security** tab, select the **Embedded LDAP** subtab.
   c. In the **Credential** field, enter the new credential.
   d. Click **Save**.
   e. Restart WebLogic Server.

   > **Note:**
   >
   > Changing the credential can affect the operation of the domain. Do not perform this step on a production server.

3. Configure a new connection to your LDAP browser using the appropriate host, port, and DN for your server instance. For example:
   - Host: `localhost`
   - Port: `7001` (`7002` if SSL is being used).
   - Base DN: `dc=`*`mydomain`* where *`mydomain`* is the name of the WebLogic domain you are using.
4. Connect to the LDAP server using the credentials that you specified in step 2.c. You cannot use anonymous bind to connect to the LDAP server.
5. Use the LDAP browser to navigate the hierarchy of the embedded LDAP server.

> **Note:**
>
> You can also view the contents of the embedded LDAP server by exporting its data and reviewing the exported file. See Exporting and Importing Information in the Embedded LDAP Server.

Exporting and Importing Information in the Embedded LDAP Server

You can export and import data from the embedded LDAP server using WLST.

See Migrating Security Data.

LDAP Access Control Syntax

The embedded LDAP server supports the IETF LDAP Access Control Model for LDAPv3. Learn how that access control is implemented within the embedded LDAP server. Apply these rules directly to entries within the directory as intended by the standard or configure and maintain them by editing the access control file (`acls.prop`).

> **Note:**
>
> The default behavior of the embedded LDAP server is to allow access only from the Administrator account in WebLogic Server. The WebLogic security providers use only the Administrator account to access the embedded LDAP server. If you are not planning to access the embedded LDAP server from an external LDAP browser or if you are planning only to use the Administrator account, you do not need to edit the `acls.prop` file and can ignore the information in this section.

The Access Control File

The access control file (`acls.prop`) maintained by the embedded LDAP server contains the complete list of access control lists (ACLs) for an entire LDAP directory. Each line in the access control file contains a single access control rule. An access control rule is made up of the following components:

- Location in the LDAP directory where the rule applies. See Access Control Location.
- Scope within that location to which the rule applies. See Access Control Scope.
- Access rights (either grant or deny). See Access Rights.
- Permissions (either grant or deny). See Attribute Permissions and Entry Permissions.
- Attributes to which the rule applies. See Attributes Types.
- Subject being granted or denied access. See Subject Types.

Example 27-1 shows a sample access control file.

Example 27-1 Sample acl.props File

```
[root]|entry#grant:r,b,t#[all]#public

ou=Employees,dc=octetstring,dc=com|subtree#grant:r,c#[all]#public:
ou=Employees,dc=octetstring,dc=com|subtree#grant:b,t#[entry]#public:
ou=Employees,dc=octetstring,dc=com|subtree#deny:r,c#userpassword#public:
ou=Employees,dc=octetstring,dc=com|subtree#grant:r#userpassword#this:
ou=Employees,dc=octetstring,dc=com|subtree#grant:w,o#userpassword,title,
description,
postaladdress,telephonenumber#this:
cn=schema|entry#grant:r#[all]#public:
```

Access Control Location

Each access control rule is applied to a given location in the LDAP directory. The location is normally a distinguished name (DN) but the special location [`root`] can be specified in the `acls.prop` file if the access control rule applies to the entire directory.

If an entry being accessed or modified on the LDAP server does not equal or reside below the location of the access control rule, the given access control rule is not evaluated further.

Access Control Scope

The following access control scopes are defined:

- Entry—An ACL with a scope of Entry is only evaluated if the entry in the LDAP directory shares the same DN as the location of the access control rule. Such rules are useful when a single entry contains more sensitive information than parallel or subentries entries.
- Subtree—A scope of Subtree is evaluated if the entry in the LDAP directory equals or ends with the location of this access control. This scope protects means the location entry and all subentries.

If an entry in the directory is covered by conflicting access control rules (for example, where one rule is an Entry rule and the other is a Subtree rule), the Entry rule takes precedence over rules that apply because of the Subtree rule.

Access Rights

Access rights apply to an entire object or to attributes of the object. Access can be granted or denied. Either of the actions `grant` or `deny` may be used when you create or update the access control rule.

Each LDAP access right is discrete. One right does not imply another right. The rights specify the type of LDAP operations that can be performed.

This section includes the following topics:

Attribute Permissions

The permissions shown in Table 27-1 apply to actions involving attributes.

Table 27-1 Attribute Permissions

Permission	Description
`r` Read	Read attributes. If granted, permits attributes and values to be returned in a Read or Search operation.
`w` Write	Modify or add attributes. If granted, permits attributes and values to be added in a Modify operation.
`o` Obliterate	Modify and delete attributes. If granted, permits attributes and values to be deleted in a Modify operation.
`s` Search	Search entries with specified attributes. If granted, permits attributes and values to be included in a Search operation.
`c` Compare	Compare attribute values. If granted, permits attributes and values to be included in a Compare operation.
`m` Make	Make attributes on a new LDAP entry below this entry.

The `m` permission is required for all attributes placed on an object when it is created. Just as the `w` and `o` permissions are used in the Modify operation, the `m` permission is used in the Add operation. The `w` and `o` permissions have no bearing on the Add operation and `m` has no bearing on the Modify operation. Since a new object does not yet exist, the a and `m` permissions needed to create it must be granted to the parent of the new object. This requirement differs from `w` and `o` permissions which must be granted on the object being modified. The `m` permission is distinct and separate from the `w` and `o` permissions so that there is no conflict between the permissions needed to add new children to an entry and the permissions needed to modify existing children of the same entry. In order to replace values with the Modify operation, a user must have both the `w` and `o` permissions.

Entry Permissions

The permissions shown in Table 27-2 apply to entire LDAP entries.

Table 27-2 Entry Permissions

Permission	Description
`a` Add	Add an entry below this LDAP entry. If granted, permits creation of an entry in the DIT subject to control on all attributes and values placed on the new entry at the time of creation. In order to add an entry, permission must also be granted to add at least the mandatory attributes.
`d` Delete	Delete this entry. If granted, permits the entry to be removed from the DIT regardless of controls on attributes within the entry.
`e` Export	Export entry and all subentries to new location. If granted, permits an entry and its subentries (if any) to be exported; that is, removed from the current location and placed in a new location subject to the granting of suitable permission at the destination. If the last RDN is changed, `Rename` permission is also required at the current location. In order to export an entry or its subentries, there are no prerequisite permissions to the contained attributes, including the RDN attribute. This is true even when the operation causes new attribute values to be added or removed as the result of the changes to the RDN.
`i` Import	Import entry and subentries from specified location. If granted, permits an entry and its subentries (if any) to be imported; that is, removed from one location and placed at the specified location (if suitable permissions for the new location are granted). When you import an entry or its subentries, the contained attributes, including the RDN attributes, have no prerequisite permissions. This is true even when the operation causes new attribute values to be added or removed as the result of the changes to RDN.
`n` RenameDN	Change the DN of an LDAP entry. Granting the Rename permission is necessary for an entry to be renamed with a new RDN, taking into account consequential changes to the DN of subentries. If the name of the superior entry is unchanged, the grant is sufficient. When you rename an entry, there are no prerequisite permissions for the contained attributes, including the RDN attributes. This is true even when the operation causes new attribute values to be added or removed as the result of the changes of RDN.
`b` BrowseDN	Browse the DN of an entry. If granted, this permission permits entries to be accessed using directory operations that do not explicitly provide the name of the entry.
`t` ReturnDN	Allows DN of entry to be disclosed in an operation result. If granted, this permission allows the distinguished name of the entry to be disclosed in the operation result.

Attributes Types

The attribute types to which an access control rule applies should be listed in the ACL where necessary. The following keywords are available:

- `[entry]` indicates the permissions apply to the entire object. This could mean actions such as delete the object, or add a child object.
- `[all]` indicates the permissions apply to all attributes of the entry.

If the keyword `[all]` and another attribute are both specified within an ACL, the more specific permission for the attribute overrides the less specific permission specified by the `[all]` keyword.

Subject Types

Access control rules can be associated with a number of subject types. The subject of an access control rule determines whether the access control rule applies to the currently connected session.

The following subject types are defined:

- `authzID`—Applies to a single user that can be specified as part of the subject definition. The identity of that user in the LDAP directory is typically defined as a DN.
- `Group`—Applies to a group of users specified by one of the following object classes:
  - `groupOfUniqueNames`
  - `groupOfNames`
  - `groupOfUniqueURLs`

  The first two types of groups contain lists of users, and the third type allows users to be included in the group automatically based on defined criteria.
- `Subtree`—Applies to the DN specified as part of the subject and all subentries in the LDAP directory tree.
- `IP Address`—Applies to a particular Internet address. This subject type is useful when all access must come through a proxy or other server. Applies only to a particular host, not to a range or subnet.
- `Public`—Applies to anyone connected to the directory, whether they are authenticated or not.
- `This`—Applies to the user whose DN matches that of the entry being accessed.

Grant/Deny Evaluation Rules

The decision whether to grant or deny a client access to the information in an entry is based on many factors related to the access control rules and the entry being protected. Throughout the decision making process, these guiding principles apply:

- More specific rules override less specific ones (for example, individual user entries in an ACL take precedence over a group entry).
- If a conflict still exists in spite of the specificity of the rule, the subject of the rule determines which rule will be applied. Rules based on an `IP Address` subject are given the highest precedence, followed by rules that are applied to a specific `AuthzID` or `This` subject. Next in priority are rules that apply to `Group` subjects. Last priority is given to rules that apply to `Subtree` and `Public` subjects.
- When there are conflicting ACL values, Deny takes precedence over Grant.
- Deny is the default when there is no access control information. Additionally, an entry scope takes precedence over a subtree scope.

Backup and Recovery

Weblogic Server provides support to recover from a corrupt embedded LDAP server file.

If any of your security realms use the Default Authentication, Authorization, Credential Mapping, or Role Mapping providers, you should maintain an up-to-date backup of the following directory tree:

domain_name/servers/*adminServer*/data/ldap

In the preceding directory, *domain_name* is the domain root directory and *adminServer* is the directory in which the Administration Server stores run-time and security data.

> **Note:**
>
> In WebLogic Server 12.2.1.3.0 and later, users are removed from the Default Authenticator LDIF templates after the users are loaded during realm initialization. Therefore, you should not delete the contents of the *domain_name*/servers directory because the data cannot be recovered. If desired, you can disable this feature by setting the system property `weblogic.security.doNotRemoveUsersFromLDIFT` to `true`. The default is `false`.

For more information about backing up the embedded LDAP server data, see the following topics:

- Back Up LDAP Repository in *Administering Server Startup and Shutdown for Oracle WebLogic Server*
- Configure the Embedded LDAP Server in *Oracle WebLogic Remote Console Online Help*.

If the embedded LDAP server file becomes corrupt or unusable, the Administration Server will generate a `NumberFormatException` and fail to start. This situation is rare but can occur if the disk becomes full and causes the embedded LDAP file to enter into an invalid state.

To recover from an unusable embedded LDAP server file, complete the following steps:

1. Change to the following directory:

   domain_name/servers/*adminServer*/data

2. Rename the embedded LDAP server file, as in the following example:

   ```
   mv ldap ldap.old
   ```

   By renaming the file, and not deleting it completely, it remains available to you for analysis and potential data recovery.

3. Start the Administration Server.

   When the Administration Server starts, a new embedded LDAP server file is created.

4. Restore any data to the new embedded LDAP server that was added since the time the WebLogic domain was created.

   If you have configured a backup of the embedded LDAP server, you can restore the backed up data by importing it. See Exporting and Importing Information in the Embedded LDAP Server.

Part VI
Configuring SSL

Learn how to configure SSL in the Oracle WebLogic Server environment.

This part contains the following chapters:

28

Overview of Configuring SSL in WebLogic Server

Learn how to configure Oracle WebLogic Server to use Secure Sockets Layer (SSL).

SSL: An Introduction

SSL provides secure connections by allowing two applications connecting over a network to authenticate each other's identity and by encrypting the data exchanged between the applications.

Authentication allows a server and optionally a client to verify the identity of the application on the other end of a network connection. Encryption makes data transmitted over the network intelligible only to the intended recipient.

SSL in WebLogic Server is an implementation of the SSL and Transport Layer Security (TLS) specifications.

> **Note:**
>
> See Table 2-1 for the supported TLS and SSL versions.

WebLogic Server supports SSL on a dedicated listen port which defaults to 7002. To establish an SSL connection over HTTP, a Web browser connects to WebLogic Server by supplying the SSL listen port and the HTTPs protocol in the connection URL, for example, `https://myserver:7002`.

Using SSL is compute intensive and adds overhead to a connection. Avoid using SSL in development environments when it is not necessary. However, always use SSL in a production environment.

This section includes the following topics:

One-Way and Two-Way SSL

SSL can be configured one-way or two-way:

- With one-way SSL, the server must present a certificate to the client, but the client is not required to present a certificate to the server. The client must authenticate the server, but the server accepts a connection from any client. One-way SSL is common on the Internet

where customers want to create secure connections before they share personal data. Often, clients will also use SSL to log on in order that the server can authenticate them.

- With two-way SSL (SSL with client authentication), the server presents a certificate to the client and the client presents a certificate to the server. WebLogic Server can be configured to require clients to submit valid and trusted certificates before completing the SSL connection.

Java Secure Socket Extension (JSSE) SSL Implementation Supported

This release of WebLogic Server uses an SSL implementation based on Java Secure Socket Extension (JSSE). JSSE is the Java standard framework for SSL and TLS and includes both blocking-IO and non-blocking-IO APIs, and a reference implementation including several commonly-trusted CAs.

The JSSE-based SSL implementation interoperates over SSL with instances of Weblogic Server version 8.1 and later that use the Certicom SSL implementation. That is, when WebLogic Server with JSSE SSL is used as either an SSL client or as the SSL server, it can communicate via SSL with instances of WebLogic Server (version 8.1 and later) that use the Certicom SSL implementation.

See Using the JSSE-Based SSL Implementation for information about using JSSE.

For complete information on JSSE, see the Java Secure Socket Extension (JSSE) Reference Guide in *Security Developer's Guide*.

> **Note:**
>
> As of WebLogic Server version 12.1.1, JSSE is the only SSL implementation that is supported. The Certicom-based SSL implementation is removed and is no longer supported in WebLogic Server.

Setting Up SSL/TLS: Main Steps

To set up SSL/TLS, you must first obtain an identity and trust for Weblogic Server and then store them using keystores. You can then configure the identity and trust keystores followed by setting SSL/TLS configuration options for the private key alias and password using the WebLogic Remote Console.

Perform the following steps to set up SSL/TLS:

1. Obtain an identity (private key and digital certificates) and trust (certificates of trusted certificate authorities) for WebLogic Server. Use the digital certificates, private keys, and trusted CA certificates provided by WebLogic Server, the CertGen utility, the keytool utility, or a reputable certificate authority (CA) to perform this step.

   > **Note:**
   >
   > If you use the CertGen utility to generate certificates, see Limitation on CertGen Usage for information about limitations on its use. Certificates generated by CertGen are for demo purposes only and should not be used in a production environment.

2. Store the identity and trust. Private keys and trusted CA certificates which specify identity and trust are stored in keystores.

> **Note:**
>
> This release of WebLogic Server supports private keys and trusted CA certificates stored in files, or in the WebLogic Keystore provider for the purpose of backward compatibility only.

3. Configure the identity and trust keystores for WebLogic Server in the WebLogic Remote Console. See Configure Keystores in *Oracle WebLogic Remote Console Online Help*.
4. Set SSL/TLS configuration options for the private key alias and password in the WebLogic Remote Console. Optionally, set configuration options that require the presentation of client certificates (for two-way SSL/TLS). See Set Up TLS in *Oracle WebLogic Remote Console Online Help*.

> **Note:**
>
> You can enable FIPS mode as described in Enabling FIPS Mode.

For information about configuring identity and trust for WebLogic Server, see the following sections:

- Obtaining and Storing Certificates for Production Environments
- Configuring Keystores with WebLogic Server

> **Note:**
>
> If your domain has two-way TLS enabled and you plan to use WLST to administer it, then you must add identity and trust properties (for the WLST client) to the user configuration file that WLST uses to connect to the server. See Connecting to Servers with Two-Way TLS Enabled in *Understanding the WebLogic Scripting Tool*.

SSL Session Behavior

WebLogic Server allows SSL sessions to be cached. Those sessions live for the life of the server. Clients that use SSL sockets directly can control the SSL session cache behavior. The SSL session cache is specific to each SSL context. All SSL sockets created by SSL socket factory instances returned by a particular SSL context can share the SSL sessions.

Clients default to resuming sessions at the same IP address and port. Multiple SSL sockets that use the same host and port share SSL sessions by default, assuming the SSL sockets are using the same underlying SSL context.

Clients that are not configured to use SSL sessions must call `setEnableSessionCreation(false)` on the SSL socket to ensure that no SSL sessions are cached. This setting only controls whether an SSL session is added to the cache; it does not stop an SSL socket from finding an SSL session that was already cached. For example, SSL

socket 1 caches the session, SSL socket 2 sets `setEnableSessionCreation` to false but it can still reuse the SSL session from SSL socket 1 because that session was put in the cache.

SSL sessions exist for the lifetime of the SSL context; they are not controlled by the lifetime of the SSL socket. Therefore, creating a new SSL socket and connecting to the same host and port used by a previous session can resume a previous session as long as you create the SSL socket using an SSL socket factory from the SSL context that has the SSL session in its cache.

By default, clients that use HTTPS URLs get a new SSL session for each URL because each URL uses a different SSL context and therefore SSL sessions can not be shared or reused. You can retrieve the SSL session by using the `weblogic.net.http.HttpsClient` class or the `weblogic.net.http.HttpsURLConnection` class. Clients can also resume URLs by sharing a SSLSocket Factory between them.

Session caching is maintained by the SSL context, which can be shared by threads. A single thread has access to the entire session cache, not just one SSL session, so multiple SSL sessions can be used and shared in a single (or multiple) thread.

You can use the `weblogic.security.SSL.sessionCache.ttl` command-line argument to modify the default server-session time-to-live for SSL session caching. See SSL in *Command Reference for Oracle WebLogic Server*. Note that the `weblogic.security.SSL.sessionCache.size` command-line argument is ignored.

29

Configuring Keystores

Learn how to configure Oracle WebLogic Server to use JKS and PKCS12 keystores for identity and trust.

For background information about identity and trust keystores, see Identity and Trust in *Understanding Security for Oracle WebLogic Server*.

For information about how to configure the Oracle OPSS Keystore Service (KSS), see Configuring Oracle OPSS Keystore Service.

About Configuring Keystores in WebLogic Server

Learn about the concepts related to the configuration and use of keystores with WebLogic Server.

About Private Keys, Digital Certificates, and Trusted Certificate Authorities

Private keys, digital certificates, and trusted certificate authorities establish and verify server identity and trust.

SSL uses public key encryption technology for authentication. With public key encryption, a public key and a *private key* are generated for a server. Data encrypted with the public key can only be decrypted using the corresponding private key and data encrypted with the private key can only be decrypted using the corresponding public key. The private key is carefully

protected so that only the owner can decrypt messages that were encrypted using the public key.

The public key is embedded in a *digital certificate* with additional information describing the owner of the public key, such as name, street address, and e-mail address. A private key and digital certificate provide *identity* for the server.

The data embedded in a digital certificate is verified by a certificate authority (CA) and digitally signed with the CA's digital certificate. Well-known certificate authorities include Entrust and Symantec Corporation. The trusted CA certificate establishes *trust* for a certificate.

An application participating in an SSL connection is authenticated when the other party evaluates and accepts the application's digital certificate. Web browsers, servers, and other SSL-enabled applications generally accept as genuine any digital certificate that is signed by a trusted CA and is otherwise valid. For example, a digital certificate can be invalidated because it has expired or the digital certificate of the CA used to sign it expired. A server certificate can be invalidated if the host name in the digital certificate of the server does not match the URL specified by the client.

Servers need a private key, a digital certificate containing the matching public key, and a certificate of at least one trusted certificate authority (CA). WebLogic Server supports private keys, digital certificates, and trusted CA certificates from the following sources:

- Private keys and digital certificates issued by a reputable CA, such as Entrust or Symantec Corporation.
- The private key and self-signed digital certificate for WebLogic Server that are created by the keytool utility.
- The demonstration digital certificates, private keys, and trusted CA certificates in the *DOMAIN_HOME*`\security`, and *JAVA_HOME*`\lib\security` directories.

  > **Note:**
  >
  > The demonstration digital certificates, private keys, and trusted CA certificates should be used in a development environment only.

- Use the digital certificates and private keys generated by the CertGen utility only for testing and demonstration purposes. These certificates should be used in a development environment only, never in a production environment.

Using Separate Keystores for Identity and Trust

When you configure SSL, you must decide how identity and trust will be stored. Although one keystore can be used for both identity and trust, Oracle recommends using separate keystores for both identity and trust because the identity keystore (holding the private key and associated digital certificate) and the trust keystore (trusted CA certificates) may have different security requirements. For example:

- For trust, you only need the certificates (non-sensitive data) in the keystore. However, for identity, you add the certificate and the private key (sensitive data) in the keystore.
- The identity keystore may be prohibited by company policy from ever being put on the corporate network, while the trust keystore can be distributed over the network.
- The identity keystore may be protected by the operating system for both reading and writing by non-authorized users, while the trust keystore only needs to be write protected.

- The identity keystore password is generally known to fewer people than the password for the trust keystore.

In general, systems within a domain have the same trust rules — they use the same set of trusted CAs — but they tend to have per-server identity. Identity requires a private key, and private keys should not be copied from one system to another. Therefore, you should maintain separate identity keystores for each system, each keystore containing only the server identity needed for that system. However, trust keystores can be copied from system to system, thus making it easier to standardize trust conventions.

Identity is more likely to be stored in hardware keystores. Trust can be stored in a file-based JDK keystore without having security issues because a trust store contains only certificates, not private keys.

Using PKCS12 Keystores in WebLogic Server

PKCS12 is an extensible, standard, and widely-supported format for storing cryptographic keys. In JDK 9, the JDK default keystore type changed from JKS to PKCS12.

The JDK default keystore type is determined by the default defined in the `keystore.type` property in the `java.security` file of your JDK installation. In JDK 8, the default was JKS. As of JDK 9, the default is PKCS12. However, you can explicitly specify the type of keystore you require. Existing keystores will not change.

Note the following as of JDK 9:

- PKCS12 keystores require a passphrase to access public certificates.
- The JDK installation provides the `cacerts` truststore. By default, WebLogic Server will use the JKS format for SSL/TLS Java Standard Trust but it also supports PKCS12.
- If you did not explicitly set the keystore type in your WebLogic Server configuration and you rely on the JDK default, when you upgrade to JDK 9 or later, the JDK default keystore type may need to be updated. In this case, if you want to continue to use JKS as the keystore type, you can set the `storetype` property in the `java.security` file to JKS. If you prefer to use PKCS12, you can convert your JKS keystores using the `-importkeystore` option of the `keytool` utility. See Converting the Default JKS Keystore for FIPS Compliance or the `keytool` utility documentation in the JDK Tool Specifications.
- For PKCS12 keystores, keytool does not support different keystore and key passwords. It uses the keystore password to persist the key. If you specify a password using the `-keypass` option and it differs from the password specified for the `-storepass` option, keytool displays a warning and ignores the keypass value.

The following table summarizes the supported keystores and defaults for WebLogic Server features and components.

Table 29-1 Keystore Type Defaults in WebLogic Server

Feature/Component	Keystore Type	Comments
SSL/TLS configuration settings for custom trust and custom identity keystores	PKCS12 or JKS	You can specify the keystore type using the configuration setting. If not specified, the JDK default keystore type in the `java.security` file is used. See Configuring Keystores.

Table 29-1 (Cont.) Keystore Type Defaults in WebLogic Server

Feature/Component	Keystore Type	Comments
Network Channel Identity Keystore	PKCS12 or JKS	You can specify the keystore type using the configuration setting. If not specified, the JDK default keystore type in the `java.security` file is used. See Configuring an Identity Keystore Specific to a Network Channel.
PKI Credential Mapping Provider Keystore	JKS (the default), or PKCS12	You can specify PKCS12 as the Keystore Type to change from the JKS default. See Configuring a PKI Credential Mapping Provider.
LDAP Authentication Provider SSL configuration	JKS or PKCS12	You can specify the keystore type to use for custom trust using the configuration setting. See Enabling an LDAP Authentication Provider for SSL.
Node Manager SSL Configuration	JKS or PKCS12	You can specify the keystore type using the configuration setting. If not specified, the JDK default keystore type in the `java.security` file is used. See Using SSL With Java-based Node Manager in *Administering Node Manager for Oracle WebLogic Server*.
Demonstration Identity and Trust Keystores	PKCS12 only	These demonstration keystores are for development use only. See Using the Demonstration Keystores
Java Standard Trust	JKS or PKCS12	• JDK 17 supplies the JDK `cacerts` in JKS format. • JDK 21 supplies the JDK `cacerts` in PKCS12 format.
JDK Keytool Utility	JKS or PKCS12	You can specify the keystore type using the command line property. If not specified, the JDK default keystore type in the `java.security` file is used. See Creating a Keystore Using Keytool
DemoCertGen Utility	PKCS12 only	The DemoCertGen utility only supports creating keystores in PKCS12 format. See Creating a Keystore Using DemoCertGen.
ImportPrivateKey Utility	JKS or PKCS12	You can specify the keystore type using the command line property. If not specified, the JDK default keystore type in the `java.security` file is used. See Creating a Keystore Using ImportPrivateKey

Configuring Keystores: Main Steps

To configure identity and trust keystores for a WebLogic Server instance being used in a production environment, complete the following steps:

1. Create the keystore to hold the server identity certificate. See Creating a Keystore.

2. Create a Certificate Signing Request (CSR), and submit it to a reputable Certificate Authority. See Generating a Certificate Signing Request. Oracle strongly recommends this step for production environments.
3. Import the identity and trust certificates returned by the CA. See Importing Certificates into the Trust and Identity Keystores.
4. Configure the trust and identity keystores with WebLogic Server. See Configuring Keystores with WebLogic Server

If you are working in a development environment where security requirements typically are less stringent, you can use the demonstration certificates included with WebLogic Server and create self-signed certificates. ***However, do not use these certificates in a production environment.*** See Using Keystores and Certificates in a Development Environment.

How WebLogic Server Locates Trust

WebLogic Server uses the following algorithm when it loads its trusted CA certificates:

1. If the trust keystore is specified by the `-Dweblogic.security.SSL.trustedCAkeystore` command-line argument in either of the following use cases, then WebLogic Server loads the trusted CA certificates from that keystore:
   - Starting a Managed Server that downloads the initial configuration from the Administration Server over SSL
   - Running a WebLogic client, such as WLST, that connects to a WebLogic Server server instance over SSL

   > **Note:**
   >
   > If, however, the Managed Server instance is started using *`DOMAIN_DIR`*`/bin/startManagedWebLogic.sh` *`managed_instance_name admin_SSL_url`*, then step 2 is not applicable to the outbound SSL connection established with the Administration Server for downloading the configuration.

2. Else if the keystore is specified in the configuration file (`config.xml`), WebLogic Server loads trusted CA certificates from the specified keystore. If the server is configured with DemoTrust, trusted CA certificates will be loaded from *`DOMAIN_HOME`*`\security\DemoTrust.p12`, *`ORACLE_HOME`*`\wlserver\server\lib\DemoTrust.jks`, and the JDK `cacerts` keystores.

   If the Oracle OPSS Keystore Service (KSS) is used, the trusted CA certificates will be loaded from `kss://system/trust`.

Creating a Keystore

You can create a PKCS12 keystore using the `DemoCertGen` utility, or a JKS or PKCS12 keystore using the `keytool`, or the `ImportPrivateKey` utilities. Oracle recommends that you keep server certificates and trusted CA certificates in separate keystores.

The following sections explain how to create a keystore. However, in practice, creating a keystore is typically done in conjunction with obtaining a server certificate for the identity keystore or importing a trusted CA certificate into the trust keystore, as explained in Obtaining and Storing Certificates for Production Environments.

- Keystore File Name Requirements

- Creating a Keystore Using DemoCertGen
- Creating a Keystore Using Keytool
- Creating a Keystore Using ImportPrivateKey

> **Note:**
>
> The preferred keystore format is JKS or PKCS12. WebLogic Server supports private keys and trusted CA certificates stored in files or in the WebLogic Keystore provider for the purpose of backward compatibility only.

Keystore File Name Requirements

When choosing a name for the keystore file:

- Do not choose a file name longer than 256 characters.
- Do not use special characters, except for an underscore (_) or hyphen (-).
- Do not use non-ASCII characters.
- Follow the operating system-specific rules for directory and file names.

Creating a Keystore Using DemoCertGen

DemoCertGen is a key and certificate management utility included in WebLogic Server. It combines functionality from the CertGen, keytool, and ImportPrivateKey utilities to configure demonstration keystores for a domain.

`DemoCertGen` creates PKCS12 keystores that are suitable for development and testing purposes only. Do not use them in production environments.

1. Change to the *`DOMAIN_HOME`*`/bin` directory of your WebLogic domain root directory.
2. Run the `setDomainEnv` script, which sets the domain-wide environment for starting and running WebLogic Server instances.
3. Run the DemoCertGen utility. Include the `-domain` argument and set its value to the domain home directory.

```
java utils.DemoCertGen -domain <DOMAIN_HOME>
```

`DemoCertGen` will generate a self-signed demo CA and then import the demonstration certificates, signed with the demo CA, into the identity and trust keystores, in the *`DOMAIN_HOME`*`/security` directory.

> **Note:**
>
> `DemoCertGen` also creates a `DemoCerts.props` file to store the passphrases for the identity and trust keystores, among other properties. This file also located in the *`DOMAIN_HOME`*`/security` directory. Do not manually edit `DemoCerts.props`.

For more information on `DemoCertGen`, see DemoCertGen in *Command Reference for Oracle WebLogic Server*.

Regenerating Demo CA and Demo Certificates using DemoCertGen

To ensure there is no lapse in service, you should regenerate your demo CA certificate and demo certificates before they expire. The demo CA certificate expires after 5 years. The demo certificates expire after 6 months.

> **Note:**
>
> You can use the `pack` and `unpack` commands to create a Managed Server template that includes the newly generated demo CA and certificates.
>
> Perform steps 1 through 4 to generate the new demo CA and certificates, then run the `pack` command with your preferred parameters. Make sure you include `-managed=true`. Then, on the remote machine, run the `unpack` command with your preferred parameters. Do not perform step 5.
>
> See Pack and Unpack Command Reference in *Creating Templates and Domains Using the Pack and Unpack Commands*.

1. On every machine with a WebLogic Server instance, make sure you back up the following files, located under *`DOMAIN_HOME`*`/security`:
   - `democacert.der`
   - `democakey.der`
   - `democert.der`
   - `demokey.der`
   - `DemoCerts.props`
   - `DemoIdentity.p12`
   - `DemoTrust.p12`
2. On the machine where the Administration Server is located, go to the *`DOMAIN_HOME`*`/bin` directory of your WebLogic domain root directory.
3. Run the `setDomainEnv` script, which sets the domain-wide environment for starting and running WebLogic Server instances.
4. Run the DemoCertGen utility. Include the `-domain` argument and set its value to the domain home directory. Typically, you will also want to include `-genIDOnly` so that only the demo certificates are regenerated and not the demo CA certificate.

   For example:

   ```
   java utils.DemoCertGen -domain DOMAIN_HOME -genIDOnly
   ```

   - Optional: If you want to regenerate *both* the demo certificates and the demo CA certificate, omit the `-genIDOnly` argument.

For example:

```
java utils.DemoCertGen -domain DOMAIN_HOME
```

5. Optional: If you have Managed Servers on other machines, then perform the following steps on each of the machines:
   a. On the Administration Server machine, copy the demo certificate files, `democacert.der` and `democakey.der`, over to *`DOMAIN_HOME`*`/security` on the Managed Server machine, replacing the existing files.
   b. Run the DemoCertGen utility with the `-genIDOnly` option.

      For example:

      ```
      java utils.DemoCertGen -domain DOMAIN_HOME -genIDOnly
      ```

Creating a Keystore Using Keytool

Keytool is a key and certificate management utility that is included in the JDK. It allows you to administer your own public/private key pairs and associated certificates for use in self-authentication (in which you authenticate yourself to other users or services) or data integrity and authentication services, using digital signatures. Keytool also allows you to cache the public keys, in the form of certificates, of your communicating peers.

When you use keytool to create a public and private key pair, keytool also creates a keystore if one does not already exist in the current directory.

> **Note:**
>
> - The default keystore type is determined by the JDK default as defined by the `keystore.type` property in the `java.security` file. As of JDK 9, the default is `pkcs12`. You can change the default by specifying the `storetype` property.
> - For PKCS12 keystores, keytool does not support different keystore and key passwords and uses the keystore password to persist the key. If you specify a password using the `-keypass` option and it differs from the password specified for the `-storepass` option, keytool displays a warning and ignores the keypass value.

To use keytool to create a JKS or PKCS12 keystore, complete the following steps:

1. Create a directory to hold the keystore. For example: *`ORACLE_HOME`*`/keystores`.
2. Change to the `bin` subdirectory of your WebLogic domain root directory. For example:

   ```
   prompt> cd DOMAIN_HOME/bin
   ```

3. Run the `setDomainEnv` script, which sets the domain-wide environment for starting and running WebLogic Server instances.
4. Change to the directory you created for the keystore and enter the following command:

   ```
   prompt> keytool -genkeypair -alias alias -keyalg RSA -keysize 2048 -dname dn -
   keystore keystore
    -storetype keystoretype
   ```

In the command, enter the following values:

- A private key alias, represented by *alias*.
- The X.500 Distinguished Name associated with the private key alias, represented by *dn*.
- The name of the keystore being created, represented by *keystore*.
- The key pair generation algorithm `RSA`.
- The type of keystore being created, either `jks` or `pkcs12`, represented by *storetype*.

When you enter the keytool command as described in the preceding steps, keytool automatically prompts you for the following:

1. The keystore password
2. The password for the private key, which is represented by its alias. Note that for PKCS12 keystores, you are not prompted for the key password.

For example:

```
prompt> keytool -genkeypair -alias server_cert -keyalg RSA -keysize 2048
 -dname "CN=server.avitek.com,OU=Support,O=Avitek,L=Reading,ST=Berkshire,C=GB" -keystore
keystore.jks -storetype jks
Enter keystore password:
Re-enter new password:
Enter key password for <server_cert>
        (RETURN if same as keystore password):
Re-enter new password:
```

Note the following from the preceding example:

- The keystore file is named `keystore.jks`.
- The private key alias is `server_cert`.
- The X.500 Distinguished Name, which consists of the WebLogic Server host and DNS domain name, is `server.avitek.com`.
- The keystore type is `jks`.

> **Note:**
>
> Make note of the private key alias and passwords you specify, and be sure to record passwords only in a safe location.

For a summary of keytool commands commonly used with WebLogic Server, see Keytool Command Summary. For details, see help for the `keytool` utility in JDK Tool Specifications.

Creating a Keystore Using ImportPrivateKey

If you have a certificate and private key, you use the ImportPrivateKey utility to create a keystore in which you can store that certificate and key.

If you used CertGen to create a private key file that is protected by a password, that password is the one required by ImportPrivateKey to extract the key from the key file and insert the key in the keystore being created.

To create a keystore using ImportPrivateKey, complete the following steps:

1. Change to the `bin` subdirectory of your WebLogic domain root directory.
2. Run the `setDomainEnv` script, which sets the domain-wide environment for starting and running WebLogic Server instances.
3. Change to the directory in which you want to create the keystore.
4. Generate the certificate and private key.

   For example, using CertGen:

   a. Enter the following command to generate the certificate file named `testcert` and the private key file named `testkey`:

   ```
   prompt> java utils.CertGen -keyfilepass mykeyfilepass -certfile testcert -
   keyfile testkey
   Generating a certificate with common name machine-name and key strength 2048
   issued by CA with certificate from CertGenCA.der file and key from
   CertGenCAKey.der file
   ```

   b. Convert the certificate from DER format to PEM format. For example:

   ```
   prompt> java utils.der2pem CertGenCA.der
   ```

   > **Note:**
   >
   > By default, the CertGen utility looks for the `CertGenCA.der` and `CertGenCAKey.der` files in the current directory, or in the *`WL_HOME`*`/server/lib` directory, as specified in the `weblogic.home` system property or the CLASSPATH.
   >
   > Alternatively, you can specify CA files on the command line. If you want to use the default settings, there is no need to specify CA files on the command line.

5. Concatenate the certificate and the Certificate Authority (CA) certificate. For example:

   ```
   prompt> cat testcert.pem CertGenCA.pem >> newcerts.pem
   ```

6. Create a new keystore and load the private key.

   For example, to create a keystore named `mykeystore` and load the private key located in the file `testkey.pem`, enter the following command:

   ```
   prompt> java utils.ImportPrivateKey -keystore mykeystore -storepass mystorepasswd -
   keyfile mykey
    -keyfilepass mykeyfilepass -certfile newcerts.pem -keyfiletestkey.pem -alias
   passalias -storetype jks
   No password was specified for the key entry
   Key file password will be used

   Imported private key testkey.pem and certificate newcerts.pem
   into a new keystore mykeystore of type jks under alias passalias
   ```

> **Note:**
>
> The default `storetype` is determined by the default for the JDK as defined by the `keystore.type` property in the `java.security` file. As of JDK 9, the default is PKCS12. You can change the default by specifying the `storetype` property.

For more information about using the ImportPrivateKey utility, see ImportPrivateKey in *Command Reference for Oracle WebLogic Server*.

Using Keystores and Certificates in a Development Environment

Learn about the tools and procedures to generate digital certificates and private keys for demonstration or testing purposes in a development environment. This information does not apply to a WebLogic Server production environment.

This section includes the following topics:

Using the Demonstration Keystores

By default, WebLogic Server is configured with two keystores, which are located in the *`DOMAIN_HOME`*`\security` directory:

- `DemoIdentity.p12`—Contains an identity certificate and demonstration private key (paired to the public key within the certificate) for WebLogic Server. This keystore contains the identity for WebLogic Server.
- `DemoTrust.p12`—Contains the CA certificate for the domain. This keystore establishes trust for WebLogic Server.

> **Note:**
>
> As of WebLogic Server 14.1.2.0.0, the demonstration keystores that are generated at domain creation, are created in PKCS12 format.

For testing and development purposes, the keystore configuration is complete. The digital certificates and trusted CA certificates in the demonstration keystores are signed by a WebLogic Server demonstration certificate authority. A unique demo CA certificate is created for each new domain. ***Do not use these demonstration keystores in a production environment.*** For information about how to configure keystores for use in a production environment, see Obtaining and Storing Certificates for Production Environments.

Creating Demonstration Certificates Using CertGen

The following sections explain the use of CertGen for creating demonstration certificates and private keys for use in a development environment:

About CertGen

The CertGen utility provides command line options to specify a CA certificate and key to be used for issuing generated certificates. The digital certificates generated by the CertGen utility by default have the host name of the machine on which they were generated, as the value for its common name field (`cn`). Beginning with WebLogic Server 14.1.1.0.0, the digital certificates by default also contain the Subject Alternative Name (SAN) extension that lists the fully-qualified DNS name as the value for the SAN extension.

Command line options let you specify values for the `cn` and other Subject domain name (`DN`) fields, such as `orgunit`, `organization`, `locality`, `state`, and `countrycode`.

Use the CertGen utility if you want to set an expiration date in the digital certificate or specify a correct host name in the digital certificate so that you can use host name verification. You can specify additional host names, or IP addresses, or both, in the SAN extension of your digital certificates by using the `-a DNS:<hostname>,IP:<ip address>` option. Optionally, you can create your certificates without the SAN extension, by using the `-nosandnshost` option at the command line. This option disables the fully-qualified DNS name and creates your certificates without the SAN extension.

The CertGen utility generates public certificate and private key files in PEM and DER formats. To view the details of the generated digital certificate on Windows platforms, double-click `.der` files in Windows Explorer

By default, the CertGen utility uses the following demonstration digital certificate and private-key files: `CertGenCA.der` and `CertGenCAKey.der`. CertGen looks for these files in the current directory, or in the *`WL_HOME`*`/server/lib directory`, as specified in the `weblogic.home` system property or the `CLASSPATH`. If you want to use these files, you do not need to specify CA files in the CertGen command; however, you can specify those CA files in the command if desired.

For complete details about the CertGen utility's syntax and arguments, see CertGen in the *Command Reference for Oracle WebLogic Server*.

Using CertGen to Create a Certificate and Private Key

To create a certificate and private key using CertGen, complete the following steps:

1. Open a command window and change to the `bin` subdirectory of your WebLogic domain root directory.
2. Run the `setDomainEnv` script. This script sets the domain-wide environment for starting and running WebLogic Server instances.
3. Optionally, change to the directory in which you want to create the certificate and private key.
4. Generate the certificate and private key using the following command:

   ```
   java utils.CertGen -keyfilepass keyfilepass -certfile cert-name -keyfile keyfile-name
   ```

   In the preceding command:

   - *`keyfilepass`* represents the password for the private key file.

- *cert-name* represents the name of the certificate.
- *keyfile-name* represents the name of the private key file.

For example, the following command generates the certificate file named `testcert` and the private key file named `testkey`:

```
prompt> java utils.CertGen -keyfilepass mykeyfilepass -certfile testcert -keyfile
testkey
Generating a certificate with common name return and key strength 2048
issued by CA with certificate from CertGenCA.der file and key from CertGenCAKey.der
file
```

CertGen Usage Notes

Note the following about using CertGen:

- By default, the CertGen utility looks for the `CertGenCA.der` and `CertGenCAKey.der` files in the current directory, or in the *WL_HOME*`/server/lib` directory, as specified in the `weblogic.home` system property or the CLASSPATH.

  Alternatively, you can specify CA files on the command line. If you want to use the default settings, there is no need to specify CA files on the command line.
- By default, the CertGen utility generates demo certificates with the SAN extension containing the fully-qualified DNS name.
  Optionally, you can create demo certificates without the SAN extension and disable the fully-qualified DNS name, by using the `-nosandnshost` command-line option.
- If you do not explicitly specify a host name with the `-cn` option, CertGen uses the JDK `InetAddress.getHostname()` method to get the host name, which CertGen inserts in the Subject common name.

  However, note that the results of the `getHostName()` method depends on the platform on which it is used. For example:

  – On some platforms, such as Solaris, this method returns a fully qualified domain name (FQDN).
  – On other platforms, such as Windows NT, this method returns a short host name.
  – On Solaris platforms, the result of `InetAddress.getHostname()` depends on how the hosts entry is configured in the `/etc/nsswitch.conf` file.

  If WebLogic Server is acting as a client and host name verification is enabled (which it is by default), you need to ensure that the host name specified in the URL matches the Subject common name in the server certificate. Otherwise, connections fail because the host names do not match.

Limitation on CertGen Usage

By default, a WebLogic Server domain is configured with the `DemoIdentity.jks` keystore, which contains a demonstration public certificate and private key for WebLogic Server. This certificate and key are created by CertGen with the default options of containing the host name in the common name field (`cn`), and the fully-qualified DNS name in the SAN (Subject Alternative Name) extension value. As a result, attempts to establish SSL connections may fail in some situations due to a host name verification exception. This section describes this limitation and provides some workarounds.

If you are using the demo certificates in a multi-server domain, Managed Server instances fail to boot if they cannot establish an SSL connection with the Administration Server. An error message similar to the following may be generated:

```
BAD_CERTIFICATE alert was
received from node-name.avitek.com - xxx.yy.zzz.yyy. Check the peer to
determine why it rejected the certificate chain (trusted CA configuration,
hostname verification). SSL debug tracing may be required to determine the
exact reason the certificate was rejected.
```

This error occurs because the host name verifier, which is enabled by default in all WebLogic domains and which is used during the SSL handshake, compares the value of the SAN extension in the certificate or the value of the `cn` field (if the certificate is created without the SAN extension) with the host name of the SSL server that accepts the SSL connection. If these names do not match, the SSL connection is dropped.

If you use the demo identity certificates in a WebLogic domain, you can use the following workarounds:

- Specify the SSL listen address of each WebLogic Server instance in a domain as the host name that appears in the certificate's `cn` field. Avoid using the fully-qualified DNS name or IP address. This workaround consists of two steps:
  1. When using the Configuration Wizard to create the WebLogic domain, specify the listen address of each WebLogic Server instance as a simple host name as it appears in the certificate's `cn` field, not as a fully-qualified DNS name or IP address. For example, if the host name in the certificate is `avitek01`, the listen address for the server instance should be specified simply as `avitek01`.
  2. At run time, when specifying the SSL listen address of a server instance, make sure the URL also matches the host name for that server as specified as the certificate's `cn` field. For example:

     ```
     https://avitek01:7002
     ```
- When starting a Managed Server instance, pass the URL of the Administration Server's SSL listening address as a parameter to the `startManagedWebLogic` script. The URL should be specified in a form that excludes the domain suffix. For example:

  ```
  C:\mydomain\bin> startManagedWebLogic.cmd https://admin01:7002
  ```
- Disable host name verification. This causes WebLogic Server to skip the verification check of ensuring that the host name in the URL to which a connection is made matches the host name in the digital certificate that the server sends back as part of the SSL connection.

  You can disable host name verification by including a command similar to the following in the `setDomainEnv` script:

  ```
  set JAVA_OPTIONS=%JAVA_OPTIONS% -
  Dweblogic.security.SSL.ignoreHostnameVerification=true
  ```

  For information about configuring host name verification, see Using Host Name Verification.

> **Note:**
>
> Oracle does not recommend using the demo certificates, or turning off host name verification, in production environments.

Using Your Own Certificate Authority

Many companies act as their own certificate authority. To use those trusted CA certificates with WebLogic Server:

1. Ensure the trusted CA certificates are in PEM format.
   - If the trusted CA certificate is in DER format, use the `der2pem` utility to convert them.
   - If the trusted CA certificate was issued by Microsoft, see Converting a Microsoft p7b Format to PEM Format.
   - If the trusted CA certificate has a custom file type, use the steps in Converting a Microsoft p7b Format to PEM Format to convert the trusted CA certificate to PEM format.
2. Create the trust keystore to hold the trusted CA certificate, as explained in Creating a Keystore.
3. Store the trusted CA certificate in the trust keystore. See Importing Certificates into the Trust and Identity Keystores.
4. Configure WebLogic Server to use the trust keystore. See Configuring Keystores with WebLogic Server.

Converting a Microsoft p7b Format to PEM Format

Digital certificates issued by Microsoft are in a format (p7b) that cannot be used by WebLogic Server. The following example converts a digital certificate in p7b (PKCS#7) format to PEM format on Windows XP:

1. In Windows Explorer, select the file (*`filename`*`.p7b`) you want to convert. Double-click on the file to display a Certificates window.
2. In the left pane of the Certificates window, expand the file.
3. Expand the Certificates folder to display a list of certificates.
4. Select a certificate to convert to PEM format. Right-click on the certificate, then choose **All Tasks** > **Export** to display the Certificate Export Wizard.
5. In the wizard, click **Next**.
6. Select the `Base-64 encoded X.509` (`.CER`) option. Then click **Next**. (`Base-64 encoded` is the PEM format.)
7. In the **File name** field, enter a name for the converted digital certificate; then click **Next**.

> **Note:**
>
> The wizard appends a `.cer` extension to the output file. The `.cer` extension is a generic extension which is appended to both base-64 encoded certificates and DER certificates. You can change the extension to `.pem` after you exit the wizard.

8. Verify that the settings are correct. If the settings are correct, click Finish; if they are not correct, click Back and make any necessary modifications.

> **Note:**
>
> For p7b certificate files that contain certificate chains, you need to concatenate the issuer PEM digital certificates to the certificate file. The resulting certificate file can be used by WebLogic Server.

Configuring Demo Certificates for Clients

To use SSL/TLS in development mode between a client and WebLogic Server, configure the demo certificates in the JVM for both the client and the server as follows:

1. Import *DOMAIN_HOME*`/security/democacert.der` into the `cacerts` keystore in the `jre/lib/security` directory of the client's JVM.

   For instructions on using the `importcert` command in `keytool`, see the `keytool` utility section in JDK Tool Specifications.
2. Import *DOMAIN_HOME*`/security/democacert.der` into the `cacerts` keystore in the `jre/lib/security` directory of the WebLogic Server's JVM.
3. Restart both the client and WebLogic Server.

> **Note:**
>
> The passphrase for the Demo Trust keystore is `DemoTrustKeyStorePassPhrase`. The passphrase for the Demo Identity keystore is stored in the *DOMAIN_HOME*`/security/DemoCerts.props` file.

Obtaining and Storing Certificates for Production Environments

To obtain a digital certificate for use in a production environment, you must generate a Certificate Signing Request (CSR) and issue it to a reputable CA. The CA returns a digital certificate that is signed with the CA's private key and that is used for establishing identity. The CA also returns the CA's signed public certificate, which is used for trust. You then import the digital certificate for identity into your identity keystore, and the CA's public certificate into the trust keystore.

The following sections explain these steps in detail:

- Generating a Certificate Signing Request
- Importing Certificates into the Trust and Identity Keystores

Generating a Certificate Signing Request

Oracle strongly recommends that all certificates used in a production environment are signed by a reputable Certificate Authority (CA). To obtain a CA-signed certificate, you must issue an individual Certificate Signing Request (CSR) for each certificate that you plan to use in that production environment.

To generate a CSR, complete the following steps:

1. Create a keystore to hold the identity of the WebLogic Server instance, if you have not already done so, as explained in Creating a Keystore.
2. Open a command window, change to the `bin` subdirectory of your WebLogic domain, and run the `setDomainEnv` script. For example, on Windows systems:

   ```
   prompt> cd DOMAIN_HOME/bin
   prompt> setDomainEnv
   ```

   In the preceding path, *`DOMAIN_HOME`* represents the WebLogic domain root directory.
3. Change to the directory that contains your keystore and create a CSR using the keytool command with the following syntax:

   ```
   keytool -certreq -v -alias alias -file certreq_file -keystore keystore
   ```

   In the preceding command syntax:

   - *`alias`* represents the private key alias specified when you created the keystore
   - *`certreq_file`* represents the name of the file that contains the CSR.
   - *`keystore`* represents the keystore.

   Note that when you enter the preceding command, you are prompted for the passwords for the keystore and the private key, which you specified when you created the keystore.
4. Submit the CSR file to a certificate authority (CA) of your choice.

The CSR file is encoded in PKCS#10 format and may look similar to the following:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBtzCCASACAQAwdzELMAkGA1UEBhMCR0IxEjAQBgNVBAgTCUJlcmtzaGlyZTEQMA
4GA1UEBxMHUmVhZGluZzEPMA0GA1UEChMGT3JhY2xlMRAwDgYDVQQLEwdTdXBwb3J0
MR8wHQYDVQQDExZtYXJzaGFsbC51ay5vcmFjbGUuY29tMIGfMA0GCSqGSIb3DQEBA
QUAA4GNADCBiQKBgQCEopgMZp1lI6jWXxb1rM1kWIc1l8bhiV/0UTcsdKzeaSHxbO
SLO3Ed9kxNWAZgXaR9f5FBlwkaRJ+IR163e64v3SplHenxHfVRaHYWPZx4KlJz/6p
Yd1fAlF0PdQm1DNoFtKmCHVk/cRuvGRpsp38l7K2mYlyQ+GxH38llS7g3owIDAQAB
oAAwDQYJKoZIhvcNAQEFBQADgYEAD/sG1+rSI76OjihHg3WezT+VIbSRJxyly9nbx
4uwXbDHh8DGgQLAXV51C9ioaMrm+dM0eygVDDMESXFxvJiYipS/pphgYt1xDBgnEH
GcNiX3BnTaLNtzYlc5eAMsmbDlpk/qOxvQiH3bKN+UKYQlBXJZWPL6FusXu2LMTrk
zsY=
-----END NEW CERTIFICATE REQUEST-----
```

> **Note:**
>
> The Certificate Request Generator servlet is deprecated. Use the keytool utility instead.

Importing Certificates into the Trust and Identity Keystores

After you submit a CSR to a CA, the CA returns the following:

- The CA's signed public certificate. (This certificate may be an intermediate certificate that is signed by a high-level CA, or it may be a self-signed (root) certificate.)

  You place this certificate into the keystore designated as the trust keystore.
- A CA-signed digital certificate for WebLogic Server. This is often referred to simply as the server certificate.

You place the server certificate into the keystore designated as the identity keystore.

- Optionally, one or more intermediate certificates that establish the chain of trust to the root CA certificate.

To import the CA-signed certificates into the trust and identity keystores, complete the following steps:

1. Open a command window, change to the `bin` subdirectory of your WebLogic domain, and run the `setDomainEnv` script. For example, on Windows systems:

   ```
   prompt> cd DOMAIN_HOME/bin
   prompt> setDomainEnv
   ```

   In the preceding path, *`DOMAIN_HOME`* represents the WebLogic domain root directory.

2. Change to the directory to hold the trust keystore and enter the following keytool command. This command creates the trust keystore, if it does not already exist, and imports the CA-signed certificate:

   ```
   keytool -importcert -file CAcert -alias CAcert-alias -keystore keystore
   ```

   In the preceding command:

   - *`CAcert`* represents the name of the CA's signed public certificate.
   - *`CAcert-alias`* represents the alias of the CA's signed public certificate.
   - *`keystore`* represents the keystore file name.

   If you currently have additional trusted CA-signed public certificates or intermediate certificates, or receive them in the future, you can add them to the preceding trust keystore using the same keytool command. For example:

   ```
   keytool -importcert -file CAcert2 -alias CAcert2-alias -keystore keystore
   ```

   If you are importing certificates that are part of a sequentially-ordered certificate path, you must import those certificates into the trust keystore in the order in which they exist in that path. If you import them in the wrong sequence, the SSL handshake when making a connection may fail. For example, consider the following certificate path:

   - Root CA certificate, `rootCA`
   - Intermediate certificate `ICA1`, which is signed by `rootCA`
   - Intermediate certificate `ICA2`, which is signed by `ICA1`

   In the preceding certificate path, you would import `rootCA` into the trust keystore first, followed by `ICA1`, then finally by `ICA2`. If these certificates are imported into the keystore in the wrong sequence,

> **Note:**
>
> Note the following:
>
> - A root CA may impose a limit on the number of intermediate certificates that may exist in a certificate path based on a root certificate issued by that CA. See Certificate Authorities in *Understanding Security for Oracle WebLogic Server*.
> - If your trust keystore does not contain the certificate of the intermediate CA that signed your server certificate, but that intermediate CA is trusted by the target of an SSL connection that you are making, the SSL connection may succeed by means of transitive trust.

3. Make a backup copy of the trust keystore.
4. Change to the directory that contains the identity keystore for WebLogic Server.
5. Import the CA-signed server certificate into your keystore using the following `keytool` command:

   ```
   keytool -importcert -v -alias alias -file servercert_file -keystore keystore
   ```

   In the preceding syntax:

   - *`alias`* represents the alias of the server certificate, which must be the same as the private key alias assigned in Step 4.)
   - *`servercert_file`* represents the name of the file that contains the CA-signed server certificate.
   - *`keystore`* represents the name of your keystore.
   - Using the `-v` option increases the amount of information displayed in the command output.

   For example, the following command imports the server certificate `server.pem` into the keystore, using the alias (`server_cert`) assigned in Step 4:

   ```
   prompt> keytool -importcert -v -alias server_cert -file server.pem -keystore
   keystore.jks
   Enter keystore password:
   Certificate reply was installed in keystore[Storing keystore.jks
   ]
   ```

6. Make a backup copy of the identity keystore.

Configuring Keystores with WebLogic Server

All private key entries in a keystore are accessed by WebLogic Server through the use of aliases, which you specify when loading private keys into the keystore. Although WebLogic Server does not use the alias to access trusted CA certificates, the keystore does require an alias when loading a trusted CA certificate into the keystore. After you have created the identity and trust keystores, you need to configure WebLogic Server to use them.

Aliases are case-insensitive: the aliases Hugo and hugo would refer to the same keystore entry. When you configure SSL, aliases for private keys are specified in the **Server Private Key Alias** field on the **Environment: Servers:** *server* page, under the **Security: SSL** tab in the WebLogic Remote Console.

See:

- Configure Keystores Using the WebLogic Remote Console in *Oracle WebLogic Remote Console Online Help*
- Configuring a Keystore Using WLST

Configuring a Keystore Using WLST

This section provides an example of using WLST to configure the identity and trust keystores for WebLogic Server. Example 29-1 does the following:

1. Connects to the Managed Server instance for which the identity and trust keystores are being configured.
2. Navigates to the MBean that corresponds to the specific server instance for which the identity and trust keystores are to be configured, `myserver`.
3. Sets the configuration rule that WebLogic Server uses to locate the identity and trust keystores, `CustomIdentityAndCustomTrust`.
4. Sets the name and location of the identity keystore file, `Identity.jks`.
5. Sets the passphrase for the identity keystore.
6. Sets the identity keystore type to `JKS`.
7. Sets the name and location of the trust keystore file, `Trust.jks`.
8. Sets the passphrase for the trust keystore.
9. Sets the trust keystore type to `JKS`.
10. Saves and activates the new keystore configuration, then disconnects from the Managed Server instance.

> **Note:**
>
> This example sets the keystore and truststore type to JKS. You can also configure PKCS12 keystores. To do so, be sure to set the `setCustomIdentityKeyStoreType()` and `setCustomTrustKeyStoreType()` properties to PKCS12.

Example 29-1 Configuring Custom Identity and Trust Keystores

```
connect('','','t3://host:port')
Please enter your username :
Please enter your password :
...
edit()
startEdit()
cd ('Servers/myserver')

cmo.setKeyStores('CustomIdentityAndCustomTrust')
cmo.setCustomIdentityKeyStoreFileName('/path/keystores/Identity.jks')
cmo.setCustomIdentityKeyStorePassPhrase('passphrase')
cmo.setCustomIdentityKeyStoreType('JKS')
cmo.setCustomTrustKeyStoreFileName('/path/keystores/Trust.jks')
cmo.setCustomTrustKeyStorePassPhrase('passphrase')
cmo.setCustomTrustKeyStoreType('JKS')
```

```
save()
activate()
disconnect()
```

Viewing Keystore Contents

Use the `keytool` command to view the contents of a keystore.

Use the following `keytool` command syntax, where *`keystore`* represents the name of the keystore you created:

```
keytool -list -v -keystore keystore
```

When you enter the preceding command, you are prompted for the keystore password. For example, the following command lists the contents of `keystore.jks`:

```
prompt> keytool -list -v -keystore keystore.jks
Enter keystore password:

Alias name: rootcacert
Creation date: Sep 13, 2010
Entry type: trustedCertEntry

Owner: CN=SSL Training CA, OU=Support, O=Avitek, L=Reading, ST=Berkshire, C=GB
Issuer: CN=SSL Training CA, OU=Support, O=Avitek, L=Reading, ST=Berkshire, C=GB
Serial number: c47f4774c2ef014c
Valid from: Fri Jan 09 10:27:18 GMT 2009 until: Mon May 26 11:27:18 BST 2036
Certificate fingerprints:
MD5: E9:24:39:56:DE:34:44:DB:46:93:45:93:8E:82:66:AC
SHA1: 17:39:92:C0:43:9B:28:F3:C2:54:55:9B:5E:97:CA:EE:71:5D:9C:26
Signature algorithm name: SHA1withRSA
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 67 57 BA 54 BB 9B C0 38 9A 71 AA 28 82 23 4B 08 gW.T...8.q.(.#K.
0010: 72 B9 FC C1 r...
]
]

#2: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
CA:true
PathLen:2147483647
]

#3: ObjectId: 2.5.29.35 Criticality=false

[CN=SSL Training CA, OU=Support, O=Avitek, L=Reading, ST=Berkshire, C=GB]
SerialNumber: [ c47f4774 c2ef014c]
]

*******************************************
*******************************************

Alias name: server_cert
Creation date: Sep 13, 2010
Entry type: PrivateKeyEntry
```

```
Certificate chain length: 2
Certificate[1]:
Owner: CN=server.avitek.com, OU=Support, O=Avitek, L=Reading, ST=Berkshire,
C=GB
Issuer: CN=SSL Training CA, OU=Support, O=Avitek, L=Reading, ST=Berkshire, C=GB
Serial number: e
Valid from: Mon Sep 13 14:02:00 BST 2010 until: Sat Sep 22 14:02:00 BST 2012
Certificate fingerprints:
MD5: CB:B8:07:32:22:B5:76:78:44:BB:94:D2:CE:EF:A3:CA
SHA1: 1E:3E:C6:BC:17:EB:43:50:19:01:0B:11:50:D8:23:60:21:B2:57:3E
Signature algorithm name: MD5withRSA
Version: 1
Certificate[2]:
Owner: CN=SSL Training CA, OU=Support, O=Avitek, L=Reading, ST=Berkshire, C=GB
Issuer: CN=SSL Training CA, OU=Support, O=Avitek, L=Readin g, ST=Berkshire, C=GB
Serial number: c47f4774c2ef014c
Valid from: Fri Jan 09 10:27:18 GMT 2009 until: Mon May 26 11:27:18 BST 2036
Certificate fingerprints:
MD5: E9:24:39:56:DE:34:44:DB:46:93:45:93:8E:82:66:AC
SHA1: 17:39:92:C0:43:9B:28:F3:C2:54:55:9B:5E:97:CA:EE:71:5D:9C:26
Signature algorithm name: SHA1withRSA
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 67 57 BA 54 BB 9B C0 38 9A 71 AA 28 82 23 4B 08 gW.T...8.q.(.#K.
0010: 72 B9 FC C1 r...
]
]

#2: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
CA:true
PathLen:2147483647
]

#3: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: 67 57 BA 54 BB 9B C0 38 9A 71 AA 28 82 23 4B 08 gW.T...8.q.(.#K.
0010: 72 B9 FC C1 r...
]

[CN=SSL Training CA, OU=Support, O=Avitek, L=Reading, ST=Berkshire, C=GB]
SerialNumber: [ c47f4774 c2ef014c]
]

*******************************************
*******************************************
```

Setting Certificate Expiry Notifications

You can set reminders to notify you when your SSL certificates are about to expire. Reminders appear in the Security Warnings Report in WebLogic Remote Console.

To set a certificate expiry notification:

1. In the WebLogic Remote Console **Edit Tree**, go to **Environment**, then **Domain**.

2. On the **Security** tab, select the **Warnings** subtab.
3. Turn on at least one of the following options: **Check Identity Certificates** or **Check Trust Certificates**.
4. In the **Check Certificates Expiration Days** field, enter a number (in days) to specify how far in advance of the certificate's expiration should the warnings begin to appear.

   By default, WebLogic Server checks that no certificates are expiring within the next 30 days.
5. In the **Check Certificates Interval Days** field, enter a number (in days) to specify how often WebLogic Server should check if the certificates are set to expire.

   By default, WebLogic Server checks daily if certificates are about to expire.
6. Click **Save**.

Replacing Expiring Certificates

You must replace an expiring certificate before it actually expires to avoid or reduce application downtime.

You can set a notification to remind you prior to its expiration. See Setting Certificate Expiry Notifications.

To replace a certificate, complete the following steps:

1. Open a command window, change to the *`DOMAIN_HOME`*`/bin` directory, and run the `setDomainEnv` script.
2. Change to the directory that contains the identity keystore that stores the certificate needing to be replaced.
3. Generate a CSR, as explained in Generating a Certificate Signing Request, using the same private key alias specified when you created the keystore for which the current expiring certificate was issued.
4. Submit the CSR to the CA that issued the original certificate. The validity date of the new certificate should be earlier than the expiration date of the current certificate. This overlap is recommended to reduce downtime.

   > **Note:**
   >
   > Steps 3 and 4 are not required if the CA already maintains the certificate request in a repository. In that case, simply ask the CA to issue a new certificate.
5. Import the newly issued certificate into the identity keystore using the alias of the private key.
6. If the new certificate is issued by a CA other than the one that issued the original certificate, you may also need to import the new CA's trusted certificate before importing the newly issued identity certificate.

Creating a Keystore: An Example

Learn how to use the keytool utility for creating a keystore and storing keys and certificates in it.

Note that this section shows only how to create one keystore. In a production environment, Oracle recommends that you have two keystores: one for trust, and another for identity, as explained in Using Separate Keystores for Identity and Trust. For complete details about each of the keytool command options shown in this section, see the help for the keytool utility at the following locations:

- **Java SE 17** - keytool in *Java Development Kit Version 17 Tool Specifications*
- **Java SE 21** - keytool in *Java Development Kit Version 21 Tool Specifications*

To create a keystore and populate it with private keys and certificates, complete the following steps:

1. Create a directory to hold the keystore; for example: *ORACLE_HOME*/keystores.
2. Run the following script, which sets the domain-wide environment for starting and running WebLogic Server instances:

   ```
   DOMAIN_HOME/bin/setDomainEnv
   ```

   In the preceding path, *DOMAIN_HOME* represents the WebLogic domain root directory.
3. Change to the directory to hold the keystore, which you created in Step 1.
4. Create the keystore using the following keytool command syntax. This command also creates a key pair (a public key and associated private key) and an alias for the private key.

   ```
   keytool -genkeypair -alias alias -keyalg RSA -keysize 2048 -dname dn -keystore
   keystore
   -storetype storetype
   ```

   In the preceding command syntax:

   - *alias* represents the private key alias.
   - *dn* represents the X.500 Distinguished Name associated with the private key alias.
   - *keystore* represents the name of the keystore being created.
   - *storetype* represents the keystore type, jks or pkcs12.

   For example:

   ```
   prompt> keytool -genkeypair -alias server_cert -keyalg RSA -keysize 2048
   -dname "CN=server.avitek.com,OU=Support,O=Avitek,L=Reading,ST=Berkshire,C=GB"
   -keystore keystore.jks
   -storetype jks
   ```

   Note the following in the preceding example:

   - server.avitek.com represents the WebLogic Server host and DNS domain name.
   - Although the keytool command includes the -storepass and -keypass options for specifying the keystore and private key passwords, respectively, Oracle recommends that you avoid using these command-line options. When you enter a keytool command that requires one or more passwords, but you omit the command-line options for passing them, you are subsequently prompted to enter them. However,

unlike passwords passed in command-line options, passwords entered in response to a prompt are not displayed in the command window and are not captured in any log.

- Make note of the private key alias and passwords you specify, and be sure to record passwords only in a safe location.

5. Make a backup copy of the keystore created in Step 4.
6. Create a Certificate Signing Request (CSR) using the following `keytool` command syntax:

   ```
   keytool -certreq -v -alias alias -file certreq_file -keystore keystore
   ```

   In the preceding command syntax:

   - *`alias`* represents the private key alias specified in Step 4.
   - *`certreq_file`* represents the name of the file that contains the CSR.
   - *`keystore`* represents the keystore created in Step 4.

   Note that when you create a CSR using the preceding command, you are prompted to enter the passwords for the keystore and the private key.

   For example, the following command creates a CSR in the file `server.csr`:

   ```
   prompt> keytool -certreq -v -alias server_cert -file server.csr -keystore
   keystore.jks
   ```

7. Submit the CSR file to a certificate authority (CA) of your choice. The CA returns:

   - A digital certificate for WebLogic Server. This certificate is signed by the CA and is often referred to simply as the server certificate.
   - The public certificate of the CA that signed your server certificate.
   - Optionally, one or more intermediate CA certificates. For example, if the CA that signed your certificate is an intermediate CA, you might also receive the public certificate of the intermediate CA that signed your CA's certificate. (If your CA's certificate was signed by a root CA, you might also receive the root certificate.)

8. In the directory you created for your keystore, save the server certificate, and also the CA certificates, in individual files. For example, the server certificate can be saved as `server.pem`, and the CA certificate as `rootCA.pem`.

   If you have an intermediate CA who also returns other intermediate certificates, save them also in your keystore directory using names such as `intermediateCA2.pem`, `intermediateCA3.pem`, and so on, to properly establish the certificate path in a way that indicates the correct sequence of that path.

9. Import the CA certificate, including any additional intermediate certificates and the root certificate if available, into your keystore using the following `keytool` command syntax:

   ```
   keytool -importcert -v -noprompt -trustcacerts -alias alias -file rootca_file -
   keystore keystore
   ```

   In the preceding syntax:

   - *`alias`* represents the alias of the root CA certificate.
   - *`rootca_file`* represents the name of the file that contains the root CA certificate.
   - *`keystore`* represents the name of your keystore.

   For example, the following command imports the root CA certificate in file `rootCA.pem` into the keystore, assigning it the alias `rootcacert`:

```
prompt> keytool -importcert -v -noprompt -trustcacerts -alias rootcacert -file
rootCA.pem -keystore keystore.jks
Enter keystore password:
Certificate was added to keystore
Storing keystore.jks
```

> **Note:**
>
> If your CA returns a certificate chain, make sure you import the certificates in the proper sequence, as explained in Importing Certificates into the Trust and Identity Keystores.

10. Import the server certificate into your keystore using the following `keytool` command syntax:

    ```
    keytool -importcert -v -alias alias -file servercert_file -keystore keystore
    ```

    In the preceding syntax:

    - *`alias`* represents the alias of the server certificate, which must be the same as the private key alias assigned in Step 4.)
    - *`servercert_file`* represents the name of the file that contains the server certificate.
    - *`keystore`* represents the name of your keystore.

    For example, the following command imports the server certificate `server.pem` into the keystore, using the alias (`server_cert`) assigned in Step 4:

    ```
    prompt> keytool -importcert -v -alias server_cert -file server.pem -keystore
    keystore.jks
    Enter keystore password:
    Certificate reply was installed in keystore[Storing keystore.jks
    ]
    ```

11. To view the contents of the keystore, use the following `keytool` command syntax, where *`keystore`* represents the name of your keystore:

    ```
    keytool -list -v -keystore keystore
    ```

Supported Formats for Identity and Trust Certificates

The PEM (Privacy Enhanced Mail) format is the preferred format for private keys, digital certificates, and trusted certificate authority (CA) certificates.

A `.pem` format file begins with this line:

```
----BEGIN CERTIFICATE----
```

and ends with this line:

```
----END CERTIFICATE----
```

A `.pem` format file supports multiple digital certificates (for example, a certificate chain can be included). The order of certificates within the file is important. The server's digital certificate should be the first digital certificate in the file, followed by the issuer certificate, and so on. Each certificate in the chain is followed by its issuer certificate. If the last certificate in the chain is the self-signed (self-issued) root certificate of the chain, the chain is considered complete. Note that the chain does not have to be complete.

When using the deprecated file-based private keys, digital certificates, and trusted CA certificates, WebLogic Server can use digital certificates in either PEM or distinguished encoding rules (DER) format.

A `.der` format file contains binary data for a single certificate. Thus, a `.der` file can be used only for a single certificate, while a `.pem` file can be used for multiple certificates.

Microsoft is often used as a CA. Microsoft issues trusted CA certificates in p7b format, which must be converted to PEM before they can be used with WebLogic Server. See Converting a Microsoft p7b Format to PEM Format.

Private key files (meaning private keys not stored in a keystore) must be in PKCS#5/PKCS#8 PEM format.

You can still use private keys and digital certificates used with other versions of WebLogic Server with this version of WebLogic Server. Convert the private key and digital certificate from distinguished encoding rules (DER) format to privacy-enhanced mail (PEM) format. See the description of the der2pem utility in "Using the WebLogic Server Java Utilities" in *Command Reference for Oracle WebLogic Server*.

After converting the files, ensure the digital certificate file has the `-----BEGIN CERTIFICATE-----` header and the `-----END CERTIFICATE-----` footer. Otherwise, the digital certificate will not work.

> **Note:**
>
> OpenSSL can add a header to the PEM certificate it generates. In order to use such certificates with WebLogic Server, everything in front of "`-----BEGIN CERTIFICATE-----`" should be removed from the certificate, which you can do with a text editor.

Obtaining a Digital Certificate for a Web Browser

The digital certification you receive for a web browser contains public information, including your name and public key, and additional information you would like authenticated by a third party, such as your E-mail address. You are required to present the digital certificate when authentication is requested.

Low-security browser certificates are easy to acquire and can be done from within the Web browser, usually by selecting the Security menu item in Options or Preferences. Go to the Personal Certificates item and ask to obtain a new digital certificate. You will be asked for some information about yourself.

As part of the process of acquiring a digital certificate, the Web browser generates a public-private key pair. The private key should remain secret. It is stored on the local file system and should never leave the Web browser's machine, to ensure that the process of acquiring a digital certificate is itself safe. With some browsers, the private key can be encrypted using a password, which is not stored. When you encrypt your private key, you will be asked by the Web browser for your password at least once per session.

> **Note:**
>
> Digital certificates obtained from Web browsers do not work with other types of Web browsers or on different versions of the same Web browser.

30

Configuring Oracle OPSS Keystore Service

Learn how to configure the Oracle OPSS Keystore Service for use with Oracle WebLogic Server. The OPSS Keystore Service makes using certificates and keys easier by providing central management and storage of keys and certificates for all servers in a domain. Configuring Keystores describes how to configure identity and trust for WebLogic Server with the default JKS keystore type.

As described in Managing Keys and Certificates with the Keystore Service in *Securing Applications with Oracle Platform Security Services*, the OPSS Keystore Service provides an alternate mechanism to manage keys and certificates for message security. You use the OPSS Keystore Service to create and maintain keystores of type KSS.

Before you proceed with configuring the OPSS Keystore Service, it is assumed that you are familiar with a basic overview of the OPSS Keystore Service, as described in Managing Keys and Certificates with the Keystore Service.

This chapter includes the following sections:

Prerequisites for Using the OPSS Keystore Service

The OPSS Keystore Service is available only with the JRF template and is not available with the default WebLogic Server configuration.

You can use the OPSS Keystore Service with WebLogic Server only if you have installed the Oracle JRF template on the WebLogic Server system as described in *Domain Template Reference* and used this template to create the domain.

Where is the OPSS Keystore Service Documented?

The OPSS Keystore Service is documented in Managing Keys and Certificates with the Keystore Service in *Securing Applications with Oracle Platform Security Services*. In particular, Managing Keys and Certificates with the Keystore Service describes how you create the KSS keystore, how to manage it, and what tools and commands are available.

This section briefly summarizes the steps you follow to configure the OPSS Keystore Service, but Managing Keys and Certificates with the Keystore Service in *Securing Applications with Oracle Platform Security Services* is the definitive source.

Configuring the OPSS Keystore Service for Demo Identity and Trust: Main Steps

You can perform the OPSS Keystore Service operations using either the Fusion Middleware Control or the Keystore Service commands with WLST. To configure the OPSS Keystore Service for demo identity and trust, you must configure the WebLogic Server instance to use demo Identity and trust, and also configure SSL for that WebLogic Server instance.

This section demonstrates the Fusion Middleware Control steps, but Managing Keys and Certificates in *Securing Applications with Oracle Platform Security Services* describes both options.

The KSS demo identity and demo trust keystores are preconfigured when you create a domain, and no additional configuration of these keystores is required.

Perform the following steps to configure an OPSS Keystore Service for demo identity and trust:

1. Configure the WebLogic Server instance to use Demo Identity and Demo Trust, as described in Configure Keystores in *Oracle WebLogic Remote Console Online Help*.
2. Configure SSL for the WebLogic Server instance, as described in Set Up TLS in *Oracle WebLogic Remote Console Online Help*.

   Remember that the WebLogic Server DefaultHostnameVerifier has been modified to accept the non-standard `DemoCertFor_<WLS Domain Name>` hostname format. Other hostname verifiers may not support this format.

   > **Note:**
   >
   > In a default JRF domain that uses KSS demo identity and trust, a Managed Server that is starting might fail to connect to the Administration Server over SSL. This failure can happen because the host name verifier cannot read the `AcceptKSSDemoCertsEnabled` setting from the SSLMBean as the MBeans are not initialized at this point. As a workaround, start the Managed Server with the `-Dweblogic.ssl.AcceptKSSDemoCertsEnabled=true` setting so that the host name verifier allows KSS demo certificates.

3. Restart WebLogic Server.

Configuring the OPSS Keystore Service for Custom Identity and Trust: Main Steps

You must configure the OPSS Keystore Service before you can use it for custom identity and trust with WebLogic Server. You can perform the OPSS Keystore Service operations using either Fusion Middleware Control or the Keystore Service commands with WLST.

This section demonstrates the Fusion Middleware Control steps, but Managing Keys and Certificates in *Securing Applications with Oracle Platform Security Services* describes both options.

Perform the following steps to configure an OPSS Keystore Service for custom identity and trust:

1. Launch Fusion Middleware Control.
2. From the **WebLogic Domain** menu, select **Security** then **Keystore**.
3. Create a keystore in the `system` stripe.
   a. Select the `system` stripe and click **Create Keystore**.

      In the Create Keystore page:

   b. Name this keystore.
   c. Set the protection type to Password.
   d. Set the password.
   e. Uncheck the Grant Permission check box.
   f. Do not specify a code base URL.
4. Select the keystore you just created and click **Manage**.

   Enter the password.
5. In the Manage Certificates page, click **Generate Keypair** to generate a private/public key pair.

   In the Generate Keypair page:

   a. Specify the alias for the key pair.
   b. Specify site-specific information as appropriate.
   c. Accept the default RSA key size if appropriate for your environment. The minimum RSA key size is 2048 bits.
   d. Specify the password.
   e. Click **OK**.
6. You have the option to use this KSS Demo CA-signed key pair as-is, or to obtain a signed certificate from a reputable vendor such as Entrust, Verisign, and so forth.

   To obtain the signed certificate from a reputable vendor, select the alias for the key pair and click **Generate CSR**. After you create a CSR, send it to your CA, which will authenticate the certificate request and create a digital certificate based on the request.

   For instructions on how to import the CA-signed certificate, see Importing a Certificate or Trusted Certificate with Fusion Middleware Control in *Securing Applications with Oracle Platform Security Services*.
7. If you do not use the preconfigured OPSS Keystore Service trust store `kss://system/trust`, you must create your own.

   > **Note:**
   >
   > Oracle recommends you use the preconfigured OPSS Keystore Service trust store.

   To create your own trust store, create another OPSS Keystore Service keystore, and import trusted certificates. For instructions on how to import trusted certificates, see Importing a Certificate or Trusted Certificate with Fusion Middleware Control in *Securing Applications with Oracle Platform Security Services.*

8. Configure the WebLogic Server instance to use KSS for Custom Identity and Trust, as described in Configure Keystores in *Oracle WebLogic Remote Console Online Help*. You specify the fully-qualified path to the keystore as the URI in the form `kss://system/keystore-name`. The keystore type is KSS.
9. Configure SSL for the WebLogic Server instance, as described in Set Up TLS in *Oracle WebLogic Remote Console Online Help*.
10. Execute the `syncKeyStores` WLST command. See Synchronizing the Local Keystore with the Security Store in *Securing Applications with Oracle Platform Security Services*
11. Restart WebLogic Server.

31

Using Host Name Verification

Learn how to configure host name verification in Oracle WebLogic Server. A host name verifier ensures the host name in the URL to which the client connects matches the host name in the digital certificate that the server sends back as part of the SSL connection. A host name verifier is useful when an SSL client (for example, WebLogic Server acting as an SSL client) connects to an application server on a remote host. It helps to prevent man-in-the-middle attacks. WebLogic Server includes two host name verifiers, and also provides the ability to create and use a custom host name verifier.

> **Note:**
>
> In releases prior to WebLogic Server 14c (14.1.1.0.0), the BEA host name verifier was also known as the default host name verifier. However, as of WebLogic Server release 14c (14.1.1.0.0), the default host name verifier is changed to the wildcard host name verifier.

This chapter includes the following sections:

Using the BEA Host Name Verifier

WebLogic Server provides two host name verifiers, the wildcard host name verfier and the BEA host name verifier. As of WebLogic Server 14c (14.1.1.0.0), the wildcard host name verifier is the default host name verifier and is configured by default.

If you are using any WebLogic Server host name verifier, host name verification passes if the host name in the certificate matches the local machine's host name, and if the URL specifies `localhost`, `127.0.0.1`, or the default IP address of the local machine.

As a function of the SSL handshake, WebLogic Server compares the common name in the SubjectDN in the SSL server's digital certificate with the host name of the SSL server used to accept the SSL connection. If these names do not match exactly, the SSL connection is dropped. The SSL client is the actual party that drops the SSL connection if the names do not match.

You can turn off host name verification or configure a custom host name verifier. Turning off host name verification leaves WebLogic Server vulnerable to man-in-the-middle attacks. Oracle recommends leaving host name verification on in production environments.

BEA host name verifier was the default host name verifier in the previous releases of WebLogic Server. To configure the BEA host name verifier, see Configuring the BEA Host Name Verifier.

> **Note:**
>
> If you are using the demo identity certificates in a multi-server domain, Managed Server instances will fail to boot if they are started using the fully-qualified DNS name of the Administration Server. For information about this limitation and suggested workarounds, see Limitation on CertGen Usage .

See Enable Host Name Verification in *Oracle WebLogic Remote Console Online Help*.

Configuring the BEA Host Name Verifier

The BEA host name verifier class name is `weblogic.security.utils.SSLWLSHostnameVerifier.DefaultHostnameVerifier`. To configure the BEA host name verifier, specify this class as a custom host name verifier in the Environment: Servers: Security: SSL page of the WebLogic Remote Console. See Enable Host Name Verification in *Oracle WebLogic Remote Console Online Help*.

Using the Wildcard Host Name Verifier

As of Oracle WebLogic Server 14c (14.1.1.0.0), the default WebLogic Server host name verifier is the wildcard host name verifier. The wildcard host name verifier is configured by default. No action is needed to use it.

In the previous releases of WebLogic Server, the BEA host name verifier was the default host name verifier. For more information about using and configuring the BEA host name verifier, see Using the BEA Host Name Verifier.

The wildcard host name verifier works the same as the BEA host name verifier; however, the wildcard host name verifier also accepts additional SSL session certificates. The wildcard host name verifier accepts the following additional SSL session certificates:

- Certificates that contain the asterisk wildcard character (`*`) in the host name that is obtained from the certificate's Subject CommonName attribute (that is, the CN domain)
- SubjectAlternativeName dnsName (SAN) certificates

This section contains the following topics:

- How the Wildcard Host Name Verifier Works
- Configuring the Wildcard Host Name Verifier

How the Wildcard Host Name Verifier Works

If the host name in the SSL session certificate contains a wildcard character that meets the following criteria, the certificate is accepted by the wildcard host name verifier:

- The host name contains at least two dot (`.`) characters.
- The host name begins with an asterisk (`*`) and does not contain any additional asterisks.
- When the asterisk (`*`) is stripped from the CN string, the remaining string must:
  - Represent the domain.
  - Include a leading dot (`.`) character.

- Be identical to the ending string of the incoming request domain.
- Not include an additional dot (.) character. (This prevents the wildcard from representing subdomains.

If the host name in the SSL session certificate does not exactly match the expected server name attribute, and the host name also cannot successfully be validated in accordance with the wildcard acceptance criteria, the wildcard host name verifier attempts to validate the SAN extensions.

The SAN extensions are obtained from the SSL session certificate. The SAN extension values are iterated using a case-insensitive match. For any iterated value, if the dnsName attribute in the certificate matches the request URL exactly or by wildcard comparison, host name verification succeeds.

Configuring the Wildcard Host Name Verifier

The wildcard host name verifier is configured by default, and is specified by the class name `weblogic.security.utils.SSLWLSWildcardHostnameVerifier`. If WebLogic Server uses a different host name verifier and if you want to restore to the default wildcard host name verifier, then specify this class as a custom host name verifier in the Environment: Servers: Security: SSL page of the WebLogic Remote Console. The wildcard host name verifier has no parameters with which it must be configured.

Using a Custom Host Name Verifier

When using a custom host name verifier, the class that implements the custom host name verifier must be specified in the CLASSPATH of WebLogic Server (when acting as an SSL client) or a standalone SSL client.

For more information about using a custom host name verifier, See Enable Host Name Verification in *Oracle WebLogic Remote Console Online Help*.

Using a Host Name Verifier on Mac OS X Platforms

If WebLogic Server is installed on a Mac OS X platform that is running in a network in which the DHCP server assigns host names, by default Mac OS X dynamically overrides the host name set on your machine, using the one assigned by DHCP. Consequently, if you have generated demo identity certificates, host name verification may fail if the host name in your certificate does not match the one that has been dynamically reassigned to your machine.

This host name reassignment can occur frequently, such as whenever the network is restarted. To use demo identity certificates with WebLogic Server on Mac OS X platforms, do one of the following:

- Disable host name verification (not recommended if operating in a production environment).
- Prior to installing WebLogic Server, set a fixed host name on your machine. Depending on your environment, you may be able to do this by changing the value of the `HOSTNAME` property in `/etc/hostconfig` from `-AUTOMATIC-` to the name you wish to assign. For example:

  ```
  HOSTNAME=mymachine.example.com
  ```

In addition, you may also verify that your desired host name is set in the file `/Library/Preferences/SystemConfiguration/preferences.plist`. Consult the Mac OS X documentation for your platform.

32

Specifying a Client Certificate for an Outbound Two-Way SSL Connection

When making an outbound two-way SSL connection, Oracle WebLogic Server, by default, uses its server certificate to establish its identity as a client. However, you can alternatively specify a separate client certificate to establish identity instead. This capability is particularly useful when WebLogic Server is acting as a client making two-way SSL connection. Learn how to specify a client certificate when making an outbound two-way SSL connection.
To use a client certificate for specifying an outbound two-way SSL connection, complete the steps described in the following sections:

- Add a Client Certificate to the Identity Keystore
- Initiate the Outbound Two-Way SSL Connection
- Restore the Use of the Server Identity Certificate

> **Note:**
>
> Switching WebLogic Server's identity to a client certificate is supported only when making an outbound two-way SSL connection. For inbound SSL connections, where Weblogic Server is acting as an SSL server, the server certificate is always used for identity.

Add a Client Certificate to the Identity Keystore

Add a client certificate to WebLogic Server's identity keystore and define the name of the alias under which the private key and public certificate are stored. This task only needs to be done once. After completing the configuration steps, the ability to use a client identity for making an outbound two-way SSL connection is always available for the current WebLogic Server instance.

To add a client certificate to the identity keystore, complete the following steps:

1. Create a client key pair (a public key and associated private key) and an alias for the private key and store it the WebLogic Server identity keystore. You can do this using the keytool utility.
2. Generate a Certificate Signing Request (CSR) and submit it to a certificate authority (CA), who returns the CA-signed client certificate. Oracle recommends using the same CA as for the server certificate so that both certificates have the same trusted root CA.
3. Store the CA-signed client certificate in the identity keystore. (If the client certificate is signed by the same CA as the server certificate, you can skip the step of storing the root CA certificate in the trust keystore because it is already there.

Initiate the Outbound Two-Way SSL Connection

Learn how to write a WLST script to initiate an outbound two-way SSL connection using the client certificate.

To initiate an outbound two-way SSL connection using the client certificate, create a WLST script that does the following:

1. Connects to the WebLogic Server instance.
2. Sets the `SSLMBean.UseServerCerts` attribute to `true`, which establishes the server identity for the outbound connection.
3. Switches to the identity of the client certificate by setting the `SSLMBean.UseClientCertForOutbound` attribute to `true`.
4. Specifies the client certificate private key passphrase, using the `SSLMBean.ClientCertPrivateKeyPassPhrase` attribute, and the client certificate keystore alias, using the `SSLMBean.ClientCertAlias` attribute.

Example 32-1 Sample WLST Script that Initiates an Outbound Two-Way SSL Connection Using a Client Identity

```
url="t3://localhost:7001"
adminUsername="weblogic"
adminPassword="password"
connect(adminUsername, adminPassword, url)
edit()
server=cmo.lookupServer('myserver')
cd('Servers')
cd('myserver')
startEdit()
cd('SSL')
cd('myserver')
ssl = server.getSSL()
ssl.setUseServerCerts(true)
ssl.setUseClientCertForOutbound(true)
ssl.setClientCertAlias("myClientCert")
ssl.setClientCertPrivateKeyPassPhrase("myClientCertPrivateKeyPassPhrase")
save()
activate()
disconnect()
exit()
```

Example 32-1 shows a WLST script that initiates an outbound two-way SSL connection using a client certificate from the identity keystore configured with WebLogic Server.

> **Note:**
>
> For clarity, this WLST example script shows the username and password in clear text. However, you should avoid entering clear-text passwords in WLST commands in general, and you should especially avoid saving on disk WLST scripts that include clear-text passwords. In these instances you should use a mechanism for passing encrypted passwords instead. See Security for WLST in *Understanding the WebLogic Scripting Tool*.

Restore the Use of the Server Identity Certificate

To restore use of the server identity certificate for outbound SSL connections, use WebLogic Remote Console or WLST to set the `SSLMBean.UseClientCertForOutbound` attribute to `false`.

In WebLogic Remote Console, go to the **Edit Tree**: **Environment**: **Servers**: *myServer*. On the **Security** tab, select the **SSL** subtab and enable **Show Advanced Fields** to display the **Use Client Cert for Outbound** option.

> **Note:**
>
> The values of the `SSLMBean.ClientCertPrivateKeyPassPhrase` and `SSLMBean.ClientCertAlias` attributes are persisted and are used the next time an outbound two-way SSL connection using a client identity is made (that is, the next time the `SSLMBean.UseClientCertForOutbound` attribute is set to `true`).

33
SSL Debugging

Learn how to enable SSL debugging in Oracle WebLogic Server. SSL debugging provides detailed information about the SSL events that occur during an SSL handshake.
This chapter includes the following sections:

About the SSL Debug Trace

The SSL debug trace provides information about the trusted certificate authorities, SSL server configuration, server identity, SSL records that were passed during the SSL handshake, and more. The SSL debugging stack trace dumps such information into a log file.

The SSL debug trace displays information about the following:

- Trusted certificate authorities
- SSL server configuration information
- Server identity (private key and digital certificate)
- The encryption strength that is allowed
- Enabled ciphers
- SSL records that were passed during the SSL handshake
- SSL failures detected by WebLogic Server (for example, trust and validity checks and the default host name verifier)
- I/O related information

SSL debugging dumps a stack trace whenever an ALERT is created in the SSL process. The types and severity of the ALERTS are defined by the Transport Layer Security (TLS) specification.

The stack trace dumps information into the log file where the ALERT originated. Therefore, when tracking an SSL problem, you may need to enable debugging on both sides of the SSL connection (on both the SSL client or the SSL server). The log file contains detailed information about where the failure occurred. To determine where the ALERT occurred, confirm whether there is a trace message after the ALERT. An ALERT received after the trace message indicates the failure occurred on the peer. To determine the problem, you need to enable SSL debugging on the peer in the SSL connection.

When tracking an SSL problem, review the information in the log file to ensure:

- The correct `config.xml` file was loaded
- The setting for domestic, or export, is correct
- The trusted certificate authority was valid and correct for this server.
- The host name check was successful
- The certificate validation was successful

> **Note:**
>
> Sev 1 type 0 is a normal close ALERT, not a problem.

Command-Line Properties for Enabling SSL Debugging

Use the command-line properties to enable debug logging within the JSSE-based SSL implementation as well as logging of the SSL calling code within WebLogic Server.

Use the following command-line properties to enable SSL debugging:

```
-Djavax.net.debug=all

-Dssl.debug=true -Dweblogic.StdoutDebugEnabled=true
```

Note the following:

- The `-Djavax.net.debug=all` property enables debug logging within the JSSE-based SSL implementation.
- The `-Dssl.debug=true` and `-Dweblogic.StdoutDebugEnabled=true` command-line properties enable debug logging of the SSL calling code within WebLogic Server.

You can include SSL debugging properties in the start script of the SSL server, the SSL client, and the Node Manager. For a Managed Server started by the Node Manager, specify this command-line argument on the Remote Start page for the Managed Server.

For information about using WebLogic logging properties with the JSSE SSL logging system, see Using Debugging with JSSE SSL.

For information about debugging utilities available for JSSE, see Debugging Utilities - Java Secure Socket Extension (JSSE) Reference Guide in *Security Developer's Guide*.

34

SSL Certificate Validation

Oracle WebLogic Server ensures that each certificate in a certificate chain was issued by a certificate authority. All X.509 V3 CA certificates used with WebLogic Server must have the Basic Constraint extension defined as CA, thus ensuring that all certificates in a certificate chain were issued by a certificate authority. By default, any certificates for certificate authorities not meeting this criteria are rejected.
This chapter describes the command-line argument that controls the level of certificate validation.

> **Note:**
>
> If WebLogic Server is booted with a certificate chain that will not pass the certificate validation, an information message is logged noting that clients could reject it.

This chapter includes the following sections:

Controlling the Level of Certificate Validation

By default, WebLogic Server rejects any certificates in a certificate chain that do not have the Basic Constraint extension defined as CA. However, you may be using certificates that do not meet this requirement or you may want to increase the level of security to conform to the IETF RFC 2459 standard. You can use a command-line argument to control this level of certificate validation.

Use the following command-line argument to control the level of certificate validation performed by WebLogic Server:

```
-Dweblogic.security.SSL.enforceConstraints=option
```

Table 34-1 describes the options for the command-line argument.

Table 34-1 Options for -Dweblogic.security.SSL.enforceConstraints

Option	Description

`strong` or `true`	Use this option to ensure that the Basic Constraints extension on the CA certificate is defined as CA.
	For example:
	`-Dweblogic.security.SSL.enforceConstraints=strong`
	or
	`-Dweblogic.security.SSL.enforceConstraints=true`
	By default, WebLogic Server performs this level of certificate validation.

`strong_nov1cas`	Functions the same as the `strong` option, described in the preceding row, with the additional constraint that X.509 version 1 CA certificates are rejected. For example: `-Dweblogic.security.SSL.enforceConstraints=strong_nov1cas`
`strict`	Use this option to ensure the Basic Constraints extension on the CA certificate is defined as CA and set to critical. This option enforces the IETF RFC 2459 standard. For example: `-Dweblogic.security.SSL.enforceConstraints=strict` This option is not the default because a number of commercially available CA certificates do not conform to the IETF RFC 2459 standard.
`strict_nov1cas`	Functions the same as the `strict` option, described in the preceding row, with the additional constraint that X.509 version 1 CA certificates are rejected. For example: `-Dweblogic.security.SSL.enforceConstraints=strict_nov1cas`
`off`	Use this option to turn off checking for the Basic Constraints extension. The rest of the certificate is still validated. For example: `-Dweblogic.security.SSL.enforceConstraints=off` Oracle does not recommend using this option in a production environment. Instead, purchase new CA certificates that comply with the IETF RFC 2459 standard. CA certificates from most commercial certificate authorities should work with the default strong option.

Accepting Certificate Policies in Certificates

WebLogic Server offers limited support for Certificate Policy Extensions in X.509 certificates. Use the `weblogic.security.SSL.allowedcertificatepolicyids` argument to provide a comma separated list of Certificate Policy IDs.

When WebLogic Server receives a certificate with a critical Certificate Policies Extension, it verifies whether any Certificate Policy is on the list of allowed certificate policies and whether there are any unsupported policy qualifiers. This release of WebLogic Server supports Certification Practice Statement (CPS) Policy qualifiers and does not support User Notice qualifiers. A certificate is also accepted if it contains a special policy `anyPolicy` with the ID 2.5.29.32.0, which indicates that the CA does not wish to limit the set of policies for this certificate.

> **Note:**
>
> The `weblogic.security.SSL.allowedcertificatepolicyids` argument is currently not supported in WebLogic Server when the JSSE-based SSL implementation is enabled.

To enable acceptance of Certificate Policies, start WebLogic Server with the following argument:

```
-Dweblogic.security.SSL.allowedcertificatepolicyids <identifier1>,<identifier2>,...
```

This argument should contain a comma-separated list of Certificate Policy identifiers for all the certificates with critical extensions that might be present in the certificate chain, back to the root certificate, in order for WebLogic Server to accept such a certificate chain.

Checking Certificate Chains

Use the WebLogic Server ValidateCertChain command-line utility to confirm whether an existing certificate chain will be rejected by WebLogic Server. The utility validates certificate chains from PEM files, PKCS-12 files, PKCS-12 keystores, and JKS keystores.

A complete certificate chain must be used with the utility. The following is the syntax for the ValidateCertChain command-line utility:

```
java utils.ValidateCertChain -file pemcertificatefilename
java utils.ValidateCertChain -pem pemcertificatefilename
java utils.ValidateCertChain -pkcs12store pkcs12storefilename
java utils.ValidateCertChain -pkcs12file pkcs12filename password
java utils.ValidateCertChain -jks alias storefilename [storePass]
```

Example of valid certificate chain:

```
java utils.ValidateCertChain -pem zippychain.pem

Cert[0]: CN=zippy,OU=FOR TESTING
ONLY,O=MyOrganization,L=MyTown,ST=MyState,C=US

Cert[1]: CN=CertGenCAB,OU=FOR TESTING
ONLY,O=MyOrganization,L=MyTown,ST=MyState,C=US

Certificate chain appears valid
```

Example of invalid certificate chain:

```
java utils.ValidateCertChain -jks mykey mykeystore

Cert[0]: CN=corba1,OU=FOR TESTING ONLY,O=MyOrganization,L=MyTown,ST=MyState,C=US

CA cert not marked with critical BasicConstraint indicating it is a CA
Cert[1]: CN=CACERT,OU=FOR TESTING ONLY,O=MyOrganization,L=MyTown,ST=MyState,C=US

Certificate chain is invalid
```

Using Certificate Lookup and Validation Providers

WebLogic Server SSL has built-in certificate validation which performs validation on the certificate chain. WebLogic Server includes two certificate lookup and validation (CLV) providers to perform additional validation on the certificate chain.

Given a set of trusted CAs, this validation:

- Verifies that the last certificate in the chain is either a trusted CA or is issued by a trusted CA.
- Completes the certificate chain with trusted CAs.
- Verifies the signatures in the chain.
- Ensures that the chain has not expired.

WebLogic Server includes two CLV providers:

- WebLogic CertPath Provider—Completes certificate paths and validates certificates using the trusted CA configured for a particular server instance, providing the same functionality as the built-in SSL certificate validation. This is configured by default.
- Certificate Registry—The system administrator makes a list of trusted CA certificates that are allowed access to the server; a certificate is valid if the end certificate is in the registry. The administrator revokes a certificate by removing it from the certificate registry, which is an inexpensive mechanism for performing revocation checking. This is not configured by default.

Alternatively, you can write a custom CertPathValidator to provide additional validation on the certificate chain. See CertPath Providers in *Developing Security Providers for Oracle WebLogic Server*.

How SSL Certificate Validation Works in WebLogic Server

Outbound SSL and two-way inbound SSL in a WebLogic Server instance receive certificate chains during the SSL handshake that must be validated. An example of two-way inbound SSL is a browser connecting to a Web application over HTTPS where the browser sends the client's certificate chain to the Web application. The inbound certificate validation setting is used for all two-way client certificate validation in the server.

Examples of WebLogic Server using outbound SSL (that is, acting as an SSL client) include:

- Connecting to the Node Manager
- Connecting to another WebLogic Server instance over the Administration port
- Connecting to an external LDAP server, such as the LDAPAuthenticator

Using any of the administration tools listed in Summary of System Administration Tools and APIs in *Understanding Oracle WebLogic Server*, you can independently configure inbound and outbound SSL certificate validation using these `SSLMBean` attributes: `InboundCertificateValidation` and `OutboundCertificateValidation`.

Legal values for both attributes are:

- `BUILTIN_SSL_VALIDATION`: Use the built-in SSL certificate validation code to complete and validate the certificate chain. That is, configure SSL to work as it has in previous releases. This is the default behavior.

- `BUILTIN_SSL_VALIDATION_AND_CERT_PATH_VALIDATORS`: Use the built-in trusted CA-based validation and the configured CertPathValidator providers to perform additional validation. That is, configure SSL to work as it has in previous releases and to do extra validation.

See:

- SSLMBean in the *MBean Reference for Oracle WebLogic Server*
- Set Up TLS in *Oracle WebLogic Remote Console Online Help*

Troubleshooting Problems with Certificate Validation

If SSL communications that worked properly in a previous release of WebLogic Server start failing unexpectedly, the likely problem is that the certificate chain is failing the validation. Determine where the certificate chain is being rejected, and decide whether to update to a certificate chain that will be accepted, or change the setting of the `-Dweblogic.security.SSL.enforceConstraints` command-line argument.

To troubleshoot problems with certificates, use one of the following methods:

- If you know where the certificate chains for the processes using SSL communication are located, use the ValidateCertChain command-line utility to check whether the certificate chains will be accepted.
- Turn on SSL debug tracing on the processes using SSL communication. The syntax for SSL debug tracing is:

```
-Dssl.debug=true -Dweblogic.StdoutDebugEnabled=true
```

> **Note:**
>
> Additional detailed debug logging may be enabled using the following command-line property:
>
> `-Djavax.net.debug=all`
>
> See Command-Line Properties for Enabling SSL Debugging.

The following message indicates the SSL failure results from problems in the certificate chain:

```
<CA certificate rejected. The basic constraints for a CA certificate were not marked
for being a CA, or were not marked as critical>
```

When you use one-way SSL, look for this error in the client log. With two-way SSL, look for this error in the client and server logs.

35

Using JCE Providers with WebLogic Server

Learn how Oracle WebLogic Server supports the use of the Jipher and JDK Java Cryptography Extension (JCE) providers.

Using the Jipher JCE Provider

The Jipher JCE provider is an Oracle developed JCE provider that is included with WebLogic Server. It is located in `jipher-jce.jar`, which is in the WebLogic Server classpath by default. It is built on top of OpenSSL and uses the OpenSSL FIPS module.

You can configure a FIPS compliant implementation of WebLogic Server by registering the Jipher JCE provider and the SunJSSE provider in the first and second positions respectively in the JDK `java.security` file as described in Enabling FIPS Mode with Jipher JCE and SunJSSE Providers.

You can register the Jipher JCE provider by customizing the `java.security` properties file in the deployment environment to modify the registered provider list or specify a system property on the command line.

- Override the default `java.security` file - note the double equal signs.

```
java -Djava.security.properties==/etc/sysconfig/jvm1.java.security
```

- Append or override parts of the `java.security` file - note the single equal sign.

```
java -Djava.security.properties=/etc/sysconfig/jvm.java.security
```

If you require support for non-FIPS compliant algorithms, you can register another JCE provider in a position below Jipher JCE, and the non-FIPS compliant algorithms will fall through to use that provider instead.

Using the JDK JCE Provider

WebLogic Server supports the use of the JDK JCE provider (`SunJCE`). The JCA framework includes an ability to enforce restrictions regarding the cryptographic algorithms and maximum cryptographic strengths available to applets/applications in different jurisdiction contexts (locations).

For more information about the features in `SunJCE`, see Java Cryptography Architecture (JCA) Reference Guide in *Security Developer's Guide*. Restrictions regarding cryptographic algorithms and cryptographic strengths are specified in the Jurisdiction Policy File Format section.

WebLogic Server will continue to control the strength of the cryptography used by the WebLogic Server Application Programming Interfaces (APIs). Client code without the appropriate domestic strength cryptography setting will only be able to use the Java SE export

strength default cryptography. On the server, WebLogic Server will enable either export or domestic strength cryptography.

36
Enabling FIPS Mode

Learn how to enable FIPS 140-2 mode in Oracle WebLogic Server.

FIPS Overview

The Federal Information Processing Standards (FIPS) 140-2 is a standard that describes U.S. Federal government requirements for sensitive but unclassified use.

You can enable a FIPS compliant (FIPS 140-2) implementation of WebLogic Server using the Jipher JCE and SunJSSE providers.

For supported versions of FIPS, see Supported FIPS Standards and Cipher Suites.

Using Jipher JCE and SunJSSE Providers

The combination of the Jipher JCE provider and the SunJSSE provider creates a FIPS-compliant implementation of WebLogic Server. Ensure Jipher JCE and SunJSSE are registered in first and second position, respectively, in the list of security providers.

Enabling FIPS Mode with Jipher JCE and SunJSSE Providers

Create a FIPS-compliant implementation of WebLogic Server with a combination of the Jipher JCE provider and the SunJSSE provider.

You can enable FIPS 140-2 mode by either creating your own `java.security` file and specifying Java options from the command line or by editing the installed JDK `java.security` file.

Enabling FIPS Mode From Java Options with Jipher

You can enable FIPS 140-2 mode with the Jipher JCE and SunJSSE providers using Java security files and specifying Java options on the command line.

1. Create your own `java.security` file. You can use the one that comes with the installed JDK as a guide.

2. Add the Jipher JCE provider as the first Java security provider listed in your `java.security` properties file. Move the rest of the providers down one position:

   ```
   security.provider.1=com.oracle.jipher.provider.JipherJCE
   security.provider.2=SunJSSE
   security.provider.3=SUN
   ```

3. Add `keystore.type=pkcs12` in your `java.security` properties file to block non-PKCS12 type keystores. If the `keystore.type=jks` system property already exists in the file, delete it.
4. Set `-Djava.security.properties` and `-Dweblogic.security.fips140strictkeystores=true` on the WebLogic Server start command line to override the default configuration in the `java.security` file and prevent WebLogic Server from using any non-PKCS12 type keystores, respectively. For `-Djava.security.properties`, specify a full file path to your custom `java.security` file.

   ```
   set JAVA_OPTIONS="-Djava.security.properties=C:\Users\user\java.security -
   Dweblogic.security.fips140strictkeystores=true"
   ```

   > **Note:**
   >
   > Use a single equal sign (=) to specify a filename if you want the java.security properties to be appended to the installed JRE security properties. Use two equal signs (==) if you want to override all the Java security properties, for instance, `-Djava.security.properties==C:\Users\user\java.security`.

5. Start WebLogic Server.

   If you are upgrading from a WebLogic Server environment that uses JKS keystores and blocking JKS keystores will cause issues, you can set `weblogic.security.fips140strictkeystores=false` and `keystore.type=jks`. However, for strict FIPS compliance, you should convert any JKS keystores instead. See Converting a Non-FIPS Compliant Keystore Using the Jipher JCE Provider for more information and conversion instructions.

Enabling FIPS 140-2 Mode From java.security

You can enable FIPS 140-2 mode from the installed JDK `java.security` file.

1. Edit the `java.security` file to add the Jipher JCE provider as the first Java security provider listed in the `java.security` properties file. Move the rest of the providers down a position:

   ```
   security.provider.1=com.oracle.jipher.provider.JipherJCE
   security.provider.2=SunJSSE
   security.provider.3=SUN
   ```

2. Add `keystore.type=pkcs12` to block non-PKCS12 type keystores. If the `keystore.type=jks` property already exists in the file, delete it.

3. Set `weblogic.security.fips140strictkeystores=true` on the WebLogic Server start command line to prevent WebLogic Server from using any non-PKCS12 type keystores.

   ```
   set JAVA_OPTIONS=-Dweblogic.security.fips140strictkeystores=true
   ```

4. Start WebLogic Server.

   If you are upgrading from a WebLogic Server environment that uses JKS keystores and blocking JKS keystores will cause issues, you can set `keystore.type=jks` in the `java.security` file and `weblogic.security.fips140strictkeystores=false` as a system property. However, for strict FIPS compliance, you should convert any JKS keystores instead. See Converting a Non-FIPS Compliant Keystore Using the Jipher JCE Provider for more information and conversion instructions.

Removing Dell JCE and Dell BSAFE JSSE Providers

Prior to WebLogic Server 14.1.2.0.0, FIPS compliance was implemented using the Dell JCE and Dell BSAFE JSSE providers. FIPS mode is now provided by the Jipher JCE and SunJSSE providers. You should remove references to the Dell providers from your WebLogic Server environment.

> **Note:**
>
> You only need to perform these steps if you are upgrading from WebLogic Server 14.1.1.0.0 or earlier and had previously modified your environment to be FIPS compliant.

1. Remove the following JAR files from the class path:
   - *MW_HOME*`/jlib/jcmFIPS.jar`
   - *MW_HOME*`/jlib/cryptoj.jar`
   - *WL_HOME*`/server/lib/sslj.jar`

   Confirm they are removed from the `PRE_CLASSPATH` environment variable as well.
2. Update the `java.security` file to remove the Dell JCE provider and the Dell BSAFE JSSE provider from the list of security providers and re-order the remaining providers.

To enable FIPS mode, see Enabling FIPS Mode with Jipher JCE and SunJSSE Providers.

Creating FIPS 140-2 Compliant Keystores

JKS or PKCS12 keystores created with the `keytool` utility and using the SunJSSE provider (the default) may not be fully FIPS compliant. To ensure that your keystores are FIPS 140-2 compliant, you can convert the keystores that you created with the SunJSSE provider by using the keytool command with the Jipher JCE provider supplied with the WebLogic Server distribution.

Although you can create a keystore with SunJSSE using FIPS-approved algorithms, if a FIPS-validated crypto implementation is not used, then it is not officially FIPS-compliant.

Also, some environments, such as Java Cloud Service configured with the Oracle Identity Cloud Integrator provider, use the default JKS keystore with CA certificates, `cacerts`. In these

environments, you must convert the JKS keystore to a FIPS compliant PKCS12 keystore using the Jipher JCE provider.

To ensure that only PKCS12 type keystores are allowed, you can set the following system property when starting WebLogic Server: `weblogic.security.fips140strictkeystores=true` and add `keystore.type=pkcs12` to the `java.security` file. If you are upgrading from a WebLogic Server environment that uses JKS, ensure that you have converted those legacy keystores to PKCS12 before, making these changes.

The following sections provide procedures for completing these steps to ensure your keystores are FIPS compliant:

- Converting a Non-FIPS Compliant Keystore Using the Jipher JCE Provider
- Converting the Default JKS Keystore for FIPS Compliance

Converting a Non-FIPS Compliant Keystore Using the Jipher JCE Provider

Using the WebLogic Server distribution classpath, you can convert a non-compliant keystore using the `keytool -importkeystore` command with the Jipher JCE provider.

To convert a non-compliant keystore using the Jipher JCE provider:

```
keytool -importkeystore -srckeystore srckeystore
-srcstoretype srcstoretype
-srcprovidername providername -destkeystore destkeystore
-deststoretype PKCS12 -destprovidername JipherJCE
-providerclass com.oracle.jipher.provider.JipherJCE
-providerpath $CLASSPATH
```

In this command, provide values for the following parameters:

- `-srckeystore` – Name of the source keystore
- `-srcstoretype` – Type of source keystore, for example `PKCS12`
- `-srcprovidername` – Name of the source keystore provider. Set to `JipherJCE` if `srcstoretype` is `PKCS12`
- `-destkeystore` - Name of the destination keystore
- `-deststoretype` – Type of destination keystore. Set to `PKCS12` for the Jipher JCE provider
- `-destprovidername` – Name of the destination keystore provider. Set to `JipherJCE` for the Jipher JCE provider
- `-providerclass` – Name of the provider class. Set to `com.oracle.jipher.provider.JipherJCE`
- `-providerpath` - Classpath for the provider

Converting the Default JKS Keystore for FIPS Compliance

FIPS 140-2 requires keystores to be in PKCS12 format using PBES2 protection; JKS keystores and PKCS12 keystores created with keytool using the Sun JSSE provider (the default) are not supported. If you are using the default JDK `cacerts` keystore, such as in a Java Cloud Service environment using the Oracle Identity Cloud Integrator provider, you need to complete the following steps to ensure FIPS compliance:

This example illustrates the steps required to convert the keystore and update the Java system properties.

1. Load the JKS keystore with the default provider and save it as a PKCS12 keystore with the Jipher JCE provider.

```
keytool -importkeystore -v
      -srckeystore $JAVA_HOME/jre/lib/security/cacerts
      -srcstoretype JKS
      -destkeystore cacerts.p12
      -deststoretype PKCS12
      -destprovidername JipherJCE
      -providerclass com.oracle.jipher.provider.JipherJCE
      -providerpath $CLASSPATH
```

2. Set the Java system properties used by the default SSL context when booting WebLogic Server. You can do this by setting the following Java options in the WebLogic Server start script as described in Specifying Java Options for a WebLogic Server Instance in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

   For example:

```
Set JAVA_OPTIONS="-Djavax.net.ssl.trustStore=/u01/jdk/jre/lib/security/
cacerts.p12 -Djavax.net.ssl.trustStoreType=PKCS12"
```

Important Considerations When Using Web Services

When using web services in FIPS 140-2 mode, there are important considerations to keep in mind.

For example:

- All certificates must have a key size length of 2048 bits.
- SHA-1 Secure Hash Algorithm Not Supported
- X509PKIPathv1 token Not Supported

SHA-1 Secure Hash Algorithm Not Supported

SHA-1 Secure Hash Algorithm is not supported in FIPS 140-2 mode. Therefore the following WS-SP `<sp:AlgorithmSuite>` values are not supported in FIPS 140-2 mode:

- Basic256
- Basic192
- Basic128
- TripleDes
- Basic256Rsa15
- Basic192Rsa15
- Basic128Rsa15
- TripleDesRsa15

As described in Using the SHA-256 Secure Hash Algorithm in *Securing WebLogic Web Services for Oracle WebLogic Server*, the WebLogic Server web service security policies support both the SHA-1 and much stronger SHA-2 (SHA-256) secure hash algorithms for hashing digital signatures. Specifically, Using the SHA-256 Policies describes which policies use the SHA-1 secure hash algorithm and their SHA-2 equivalents.

FIPS 140-2 mode requires an Extended Algorithm Suite when digital signatures are used. See Using the Extended Algorithm Suite (EAS) in *Securing WebLogic Web Services for Oracle WebLogic Server*.

If you enable FIPS 140-2 mode, change the `<sp:AlgorithmSuite>` element in the Security policy to one of the following supported `<sp:AlgorithmSuite>` values as described in Using the SHA-256 Secure Hash Algorithm:

- Basic256Sha256
- Basic192Sha256
- Basic128Sha256
- Basic256Exn256
- Basic192Exn256
- Basic128Exn256
- TripleDesSha256
- TripleDesExn256
- Basic256Sha256Rsa15
- Basic192Sha256Rsa15
- Basic128Sha256Rsa15
- Basic256Exn256Rsa15
- Basic192Exn256Rsa15
- Basic128Exn256Rsa15
- TripleDesSha256Rsa15
- TripleDesExn256Rsa15

For example, to edit an existing Basic256 Algorithm Suite to an EAS Algorithm Suite, then change the policy from

```
<sp:AlgorithmSuite>
        <wsp:Policy>
             <sp:Basic256/>
        </wsp:Policy>
</sp:AlgorithmSuite>
```

to

```
<sp:AlgorithmSuite>
         <wsp:Policy>
             <orasp:Basic256Exn256 xmlns:orasp="http://schemas.oracle.com/ws/2006/01/
securitypolicy"/>
         </wsp:Policy>
</sp:AlgorithmSuite>
```

X509PKIPathv1 token Not Supported

The X509PKIPathv1 token is not supported for FIPS 140-2 mode in this release of WebLogic Server. If you use the X509PKIPathv1 token in a custom policy, change the policy to use the PKCS7 token instead.

Specifically, the following two policy assertions are not supported in FIPS 140-2 mode in this release of WebLogic Server:

- `<sp:WssX509PkiPathV1Token10/>`
- `<sp:WssX509PkiPathV1Token11/>`

If you use these two policy assertions, change them to the following two assertions instead:

- `<sp:WssX509Pkcs7Token10/>`
- `<sp:WssX509Pkcs7Token11/>`

For example, if the policy has the following assertion in the custom policy:

```
<wsp:Policy>
   <sp:X509Token sp:IncludeToken=". . .">
         <wsp:Policy>
               <sp:WssX509PkiPathV1Token10/>
         </wsp:Policy>
   </sp:X509Token>
</wsp:Policy>
```

replace it with the following policy assertion:

```
<wsp:Policy>
   <sp:X509Token sp:IncludeToken=". . .">
         <wsp:Policy>
               <sp:WssX509Pkcs7Token10/>
         </wsp:Policy>
   </sp:X509Token>
</wsp:Policy>
```

Or, if the policy has the following assertion in the custom policy:

```
<wsp:Policy>
   <sp:X509Token sp:IncludeToken=". . .">
         <wsp:Policy>
              <sp:RequireThumbprintReference/>
              <sp:WssX509PkiPathV1Token11/>
         </wsp:Policy>
   </sp:X509Token>
</wsp:Policy>
```

replace it with the following assertion:

```
<wsp:Policy>
   <sp:X509Token sp:IncludeToken=". . .">
         <wsp:Policy>
               <sp:RequireThumbprintReference/>
               <sp:WssX509Pkcs7Token11/>
         </wsp:Policy>
   </sp:X509Token>
</wsp:Policy>
```

37

Specifying the SSL/TLS Protocol Version

Learn how to configure Oracle WebLogic Server to limit the lowest supported versions of SSL and TLS that are enabled for SSL connections.

About the SSL Version Used in the Handshake

At the start of the SSL handshake, the SSL peers determine the highest protocol version both peers support. However, you can configure Weblogic Server to limit the lowest supported versions of SSL and TLS that are enabled for SSL connections by using the command-line utility.

To specify the SSL and TLS versions enabled for the SSL handshake, you can set either of the following system properties in the command-line argument that starts WebLogic Server:

- `weblogic.security.SSL.protocolVersion`
- `weblogic.security.SSL.minimumProtocolVersion`

Note that WebLogic Server supports the JSSE-based SSL implementation only. See Using the JSSE-Based SSL Implementation.

Using the weblogic.security.SSL.protocolVersion System Property

You can specify which protocol, SSL or TLS, is used when making SSL connections. Some circumstances such as compatibility, SSL performance, and security requirements make TLS the better choice.

Use the `weblogic.security.SSL.protocolVersion` system property as a command-line argument when starting WebLogic Server to specify which protocol is used for SSL connections. The following command-line arguments can be specified so that WebLogic Server supports only SSL v3.0 or TLS connection.

- `-Dweblogic.security.SSL.protocolVersion=SSL3`—Only SSL v3.0 messages are sent and accepted. Attempts by clients to establish connections with a prior SSL version will be denied by WebLogic Server, with a denial message returned to the client.

> **Note:**
>
> SSLv3 may be disabled by default in certain JDK updates by the underlying JSSE provider. If so, then enabling SSLv3 in WebLogic Server may not take effect and you will see runtime errors for SSL connections.
>
> Oracle strongly recommends that you do not use SSLv3. If you want to use SSLv3, then you must remove SSLv3 from the `jdk.tls.disabledAlgorithms` JDK setting specified in the `java.security` file, and then enable SSLv3 in WebLogic Server.

- `-Dweblogic.security.SSL.protocolVersion=TLS1`— This property value enables any protocol starting with "TLS" for messages that are sent and accepted; for example, TLS v1.0, TLS v1.1, TLS v1.2, and TLS v1.3.

> **Note:**
>
> Support for TLS v1.0 and v1.1 is deprecated. Oracle strongly recommends that you do not use TLS v1.0 and v1.1. In addition, these versions may be disabled by default in certain JDK updates by the underlying JSSE provider.

- `-Dweblogic.security.SSL.protocolVersion=ALL`—This is the default behavior. If ALL is selected, the default depends on the JSSE provider and JDK version. For the supported protocol version table for Sun JSSE, see the *SunJSSE Provider* section in Java SE Security Developer's Guide. .

Note the following:

- The SSL v3.0 and TLS v1 protocols can not be interchanged. Use only the TLS v1 protocol if you are certain all desired SSL clients are capable of using the protocol.
- Not setting the `weblogic.security.SSL.protocolVersion` system property enables the SSLv3Hello, SSLv3, and TLS v1 protocols. In addition, for JSSE, all versions starting with "TLS" are also enabled.
- If you set valid, supported protocols for the `weblogic.security.SSL.minimumProtocolVersion` system property, the protocol value you set for `weblogic.security.SSL.protocolVersion` is ignored.

> **Note:**
>
> - If you specify the `TLS1` or `ALL` value in this system property, all versions of TLS v1 supported by the SSL provider are enabled for use in SSL connections. The JSSE-based implementation supports TLS v1.0, TLS v1.1, TLS v1.2, and TLS v1.3.
> - TLS v1.2 is the default minimum protocol version configured in WebLogic Server. WebLogic Server logs a warning if the TLS version is set below 1.2.

Using the weblogic.security.SSL.minimumProtocolVersion System Property

In a production environment, Oracle recommends TLS v1.2, or later (if supported by the underlying JSSE provider), for sending and receiving messages in an SSL connection. To control the minimum versions of SSL v3.0 and TLS v1 that are enabled for SSL connections, set the `weblogic.security.SSL.minimumProtocolVersion=`*`protocol`* as a command line option when starting WebLogic Server.

> **Note:**
>
> TLS v1.2 is the default minimum protocol version configured in WebLogic Server. WebLogic Server logs a warning if the TLS version is set below 1.2.

This system property accepts one of the following values for *`protocol`*:

Value	Description
`SSLv3`	Specifies SSL v3.0 as the minimum protocol version enabled in SSL connections.
`TLSv1`	Specifies TLS v1.0 as the minimum protocol version enabled in SSL connections. **Note:** By default, WebLogic Server uses TLS v1.2 as the minimum protocol version and logs a warning if the TLS version is set below 1.2.
`TLSv`*`x.y`*	Specifies TLS v*`x.y`* as the minimum protocol version enabled in SSL connections, where: • *`x`* is an integer between 1 and 9, inclusive • *`y`* is an integer between 0 and 9, inclusive For example, `TLSv1.2`. WebLogic Server logs a warning if the TLS version is set below 1.2.

The specific protocols that are enabled by each of the values you can specify for the `weblogic.security.SSL.minimumProtocolVersion` system property depend upon the SSL implementation with which WebLogic Server is configured.

Protocols Enabled with the JSSE-Based SSL Implementation identifies these protocols for the JSSE-based SSL implementation available in WebLogic Server:

> **Note:**
>
> The `weblogic.security.SSL.minimumProtocolVersion` system property cannot take effect if the `jdk.tls.client.protocols` JDK system property is specified.

Protocols Enabled with the JSSE-Based SSL Implementation

When WebLogic Server is configured to use the JSSE-based SSL implementation and you specify a minimum protocol version using the `weblogic.security.SSL.minimumProtocolVersion` system property, the specific SSL and TLS

protocols that are enabled depend on the protocols that are supported in the SSL implementation, as follows:

- If the particular minimum protocol version you specify is supported, WebLogic Server enables that protocol version *and* all later protocol versions that are supported.

  For example:

If you specify . . .	and the JSSE-based SSL implementation supports . . .	the following protocols are enabled
TLSv1	SSLv3	TLSv1
	TLSv1	TLSv1.1
	TLSv1.1	TLSv1.2
	TLSv1.2	TLSv1.3
	TLSv1.3	

- If the particular minimum protocol version you specify is *not* supported, Weblogic Server enables the next lower protocol and all later protocols that are supported. Note that the lowest protocol will be limited to SSLv3.

  For example:

If you specify . . .	and the JSSE-based SSL implementation supports . . .	the following protocols are enabled
TLSv1	SSLv3	SSLv3
	TLSv1.1	TLSv1.1
	TLSv1.2	TLSv1.2
	TLSv1.3	TLSv1.3

- If the exact minimum protocol you specify is *not* supported, and no older (lower) protocol is supported that is SSLv3 or higher, WebLogic Server enables all newer (higher) supported versions. This case usually applies when SSLv3 is set as the minimum.

  For example:

If you specify . . .	and the JSSE-based SSL implementation supports . . .	the following protocols are enabled
SSLv3	TLSv1	TLSv1
	TLSv1.1	TLSv1.1
	TLSv1.2	TLSv1.2
	TLSv1.3	TLSv1.3

- If the particular minimum protocol you specify is invalid, WebLogic Server enables SSLv3 *and* all later protocol versions that are supported.

  For example:

If you specify . . .	and the JSSE-based SSL implementation supports . . .	the following protocols are enabled
TSLv0	SSLv3	SSLv3
	TLSv1	TLSv1
	TLSv1.1	TLSv1.1
	TLSv1.2	TLSv1.2
	TLSv1.3	TLSv1.3

Note:

- Support for TLS v1.0 and v1.1 is deprecated. Oracle strongly recommends that you do not use TLS v1.0 and v1.1. In addition, these versions may be disabled by default in certain JDK updates by the underlying JSSE provider.
- Due to its vulnerability to security attacks, SSLv3 may be disabled by default in certain JDK updates by the underlying JSSE provider. If so, then setting SSLv3 using the `weblogic.security.SSL.minimumProtocolVersion` system property may not take effect. Oracle strongly recommends that you do not use SSLv3. If you want to use SSLv3, then you must remove SSLv3 from the `jdk.tls.disabledAlgorithms` JDK setting specified in the `java.security` file, and then enable SSLv3 in WebLogic Server.

Using the weblogic.security.ssl.sslcontext.protocol System Property

For some JSSE providers, there is a correlation between the `javax.net.ssl.SSLContext` algorithm and the initially enabled SSL/TLS protocols. WebLogic Server includes a `weblogic.security.ssl.sslcontext.protocol` system property that provides the ability to specify a custom `javax.net.ssl.SSLContext` algorithm for your JSSE provider.
The default protocol setting used with the Oracle JDK JSSE provider is `TLS`. Some vendors interpret the protocol parameter differently and you may need to change the setting. Refer to the vendor-specific documentation for the correlations between the `javax.net.ssl.SSLContext` setting and the enabled SSL/TLS protocols.

Note:

When using the IBM JSSE provider, WebLogic Server attempts to select a `javax.net.ssl.SSLContext` algorithm equivalent to the default `TLS`.

If a custom `javax.net.ssl.SSLContext` algorithm is required for use by WebLogic Server, you can set the system property at the command line as follows:

```
-Dweblogic.security.ssl.sslcontext.protocol=protocol
```

The *protocol* parameter is a key for selecting a specific `javax.net.ssl.SSLContext` algorithm. When set, it overrides the default value. Standard supported values are `SSL`, `SSLv3`, `TLS`, `TLSv1`, `TLSv1.1`, `TLSv1.2`, and `TLSv1.3`. WebLogic Server does not support `SSLv2`. Alternatively, you can set the property to a custom value supported by the underlying JSSE provider, however it may affect which SSL/TLS protocol versions are enabled in the TLS connections. See:

- SSLContext Algorithms in *Java Security Standard Algorithm Names* for JDK 17
- SSLContext Algorithms in *Java Security Standard Algorithm Names* for JDK 21

Note:

- Support for TLS v1.0 and v1.1 is deprecated. Oracle strongly recommends that you do not use TLS v1.0 and v1.1. In addition, these versions may be disabled by default in certain JDK updates by the underlying JSSE provider.
- SSLv3 may be disabled by default in certain JDK updates by the underlying JSSE provider. Oracle strongly recommends that you do not use SSLv3.

38
Using the JSSE-Based SSL Implementation

Learn how to use the JSSE-based SSL implementation and understand the supported cipher suites and the differences in the Certicom-based implementation of JSSE.

> **Note:**
>
> Note the following:
>
> - As of WebLogic Server version 12.1.1, JSSE is the only SSL implementation that is supported. The Certicom-based SSL implementation is removed and is no longer supported in WebLogic Server.
> - SHA-2 signed certificates are supported in the JSSE SSL implementation provided in WebLogic Server.
> - Although JSSE supports Server Name Indication (SNI) in its SSL implementation, WebLogic Server does not support SNI.

This chapter includes the following sections:

System Property Differences Between the JSSE-Based and Certicom SSL Implementations

Learn the differences in how the JSSE and Certicom SSL implementations handle the WebLogic security system properties.

Table 38-1 System Properties Differences

System Property	JSSE Applicability	Description
`weblogic.security.SSL.ignoreHostnameVerification`	This property continues to work and is not affected by the JSSE integration.	Does not verify the hostname in the URL to the hostname in the certificate.
`weblogic.ReverseDNSAllowed`	This property continues to work and is not affected by the JSSE integration.	If set to true then use reverse DNS lookup to figure out if `urlhostname` is a loopback address ("localhost" or "127.0.0.1", or the IPV6 equivalent.

Table 38-1 (Cont.) System Properties Differences

System Property	JSSE Applicability	Description
`weblogic.security.SSL.trustedCAKeyStore`	This property continues to work and is not affected by the JSSE integration.	Loads the trusted CA certificates from that keystore.
`weblogic.security.SSL.verbose`	Use this property in combination with `javax.net.debug=all` to get verbose debug output from the SSL calling code and the JSSE-based implementation.[1]	For additional SSL debugging when `-Dssl.debug=true` is used.
`ssl.debug=true`	Use this property in combination with `javax.net.debug=ssl` to get debug output from the SSL calling code and the JSSE-based implementation.[1]	Displays SSL debug information to the console or logs. This property is for the calling WebLogic code. The JSSE-based SSL implementation has its own logging system, which is activated by the `javax.net.debug` property. **Note:** You can set JSSE logging (`javax.net.debug`) independently of WebLogic SSL logging (`ssl.debug`).
`weblogic.security.SSL.disableJsseCipherSuiteAliases=true|false`	The default is `false`.	Disables the conversion of Certicom cipher suite names to `SunJSSE` cipher suite names, where applicable. By default, Certicom cipher suite names are converted to JSSE cipher suite names when JSSE is used for SSL. For a list of Certicom cipher suite names and their `SunJSSE` equivalents, see Table 38-2.
`weblogic.security.SSL.ignoreHostnameVerify`	This property continues to work and is not affected by the JSSE integration.	See `weblogic.security.SSL.ignoreHostnameVerification`
`weblogic.security.SSL.HostnameVerifier=`*`classname`*	This property continues to work and is not affected by the JSSE integration.	Specifies the class name of a custom hostname verification class.
`weblogic.security.SSL.protocolVersion=`*`protocol`*	This property continues to work and is not affected by the JSSE integration. The supported protocol values are mapped to the corresponding protocols supported by JSSE.	See Specifying the SSL/TLS Protocol Version.

Table 38-1 (Cont.) System Properties Differences

System Property	JSSE Applicability	Description
One of the following: • `weblogic.security.SSL.allowUnencryptedNullCipher` • `SSLMBean.SetAllowUnencryptedNullCipher(boolean)` • `weblogic.security.disableNullCipher`	`SunJSSE` supports the following two null ciphers, but they are not enabled by default: • `SSL_RSA_WITH_NULL_MD5` • `SSL_RSA_WITH_NULL_SHA` If this setting is enabled, these two null ciphers are added to the cipher list.	By default, this control is not set and the use of a null cipher is not allowed on the server. In such a configuration, if the SSL clients want to use the null cipher suite (by indicating `SSL_RSA_WITH_NULL_MD5` as the only supported cipher suite), the SSL handshake will fail. If you set this control, the null cipher suite (for example, `SSL_RSA_WITH_NULL_MD5`) is added to the list of supported cipher suites by the server. The SSL connection has a chance to use the null cipher suite if the client wants to do so. If the null cipher suite is used, the message will be unencrypted. **Caution**: Do not set this control in a production environment unless you are aware of the implications and consequences of doing so.
`weblogic.security.SSL.enforceConstraints=`*option*	`Off` is not supported, but other options are supported.	Ensures that the Basic Constraints extension on the CA certificate is defined as CA. See Controlling the Level of Certificate Validation.
`weblogic.security.SSL.allowedcertificatepolicyids`	Not supported.	WebLogic Server offers limited support for Certificate Policy Extensions in X.509 certificates. See Accepting Certificate Policies in Certificates.
`weblogic.security.SSL.nojce`	Not supported.	See Setting Up SSL/TLS: Main Steps.

[1] This WebLogic system property is applicable to both the Certicom and JSSE-based SSL implementations. However, for JSSE, this property affects only the SSL calling code, not the JSSE-based implementation. For more information about the `javax.net.debug` system property and debugging the JSSE-based SSL implementation, see Debugging Utilities - Java Secure Socket Extension (JSSE) Reference Guide in *Security Developer's Guide*.

Cipher Suites

Learn about the cipher suites supported by WebLogic Server, using anonymous ciphers, and setting cipher suites.

To set cipher suites, use WebLogic Remote Console or WLST. See Set Cipher Suites in *Oracle WebLogic Remote Console Online Help* or Setting Cipher Suites Using WLST: An Example.

This topic includes the following sections:

List of Supported Cipher Suites

For a list of the set of cipher suites supported by the JDK default JSSE provider, `SunJSSE`, see:

- JDK 17: The SunJSSE Provider in *Security Developer's Guide*
- JDK 21: The SunJSSE Provider in *Security Developer's Guide*

Deprecated Cipher Suites

Per Oracle security guidelines, the TLS cipher suites that are prefixed with `TLS_RSA_` or contain `_CBC_` are deprecated and are disabled by default. These disabled cipher suites are weak and do not provide sufficient security for your system. However, if necessary for your environment, you can enable these TLS cipher suites using any of the following methods:

- Set the `ExcludedCiphersuites` attribute on the `weblogic.management.configuration.SSLMBean` MBean to an array that contains just one empty string. For example `new String[]{""}`.
- Set the `MinimumTLSProtocolVersion` attribute on the `weblogic.management.configuration.SSLMBean` MBean to `TLSv1.1` or earlier.
- Set the system property `-Dweblogic.security.SSL.minimumProtocolVersion` to `TLSv1.1` or earlier in the Java command that starts WebLogic Server.
- Set the system property `-Dweblogic.security.SSL.protocolVersion` in the Java command that starts WebLogic Server. See Using the weblogic.security.SSL.protocolVersion System Property.

Backward Compatibility of Supported Cipher Suites

For backward compatibility, the JSSE-based SSL implementation accepts Certicom cipher suite names for cipher suites that are compatible with `SunJSSE`. The Certicom cipher suite names are converted for you to `SunJSSE` equivalents, usually replacing the "TLS_" prefix with "SSL_", as shown in Table 38-2.

Please keep the following in mind as you consider backward compatibility with Certicom cipher suites:

- With JSSE, the cipher suites selected by default are stronger as compared to Certicom SSL and have slower performance. The security policies in your environment typically set the requirements for the cipher suites that must be used. However, for highly secure environments, using the strongest available cipher that provides acceptable performance is recommended.
- For operations where enabled or supported cipher suites are returned, both the Certicom and `SunJSSE` names of the cipher suites are returned. (Note that the

`weblogic.security.SSL.disableJsseCipherSuiteAliases=true` property, described in Table 38-1, disables this behavior.)

- For operations where you specify enabled cipher suites, you can use either the equivalent Certicom cipher suite names, or the `SunJSSE` names. The Certicom cipher suites, and their `SunJSSE` equivalents, are listed in Table 38-2. (Oracle does not encourage future use of Certicom cipher suite names.)
- The `_DSS_` cipher suites requires certificates signed with DSS, the Digital Signature Standard defined by NIST FIPS Pub 186. DSA is the key generation scheme as described in FIPS 186.
- The `_anon_` cipher suites are disabled by default. To enable them, you can use WebLogic Remote Console or WLST. See Set Cipher Suites in *Oracle WebLogic Remote Console Online Help* or Setting Cipher Suites Using WLST: An Example.
- The TLS DES cipher suites have been disabled at the JSSE provider level by default. See the *Oracle JRE and JDK Cryptographic Roadmap* at `https://java.com/en/jre-jdk-cryptoroadmap.html`.

Using Anonymous Ciphers

The following anonymous ciphers are not supported out-of-the-box in the JSSE-based WebLogic SSL implementation in WebLogic Server:

- `TLS_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `TLS_DH_anon_WITH_RC4_128_MD5`
- `TLS_DH_anon_WITH_DES_CBC_SHA`
- `TLS_DH_anon_EXPORT_WITH_RC4_40_MD5`
- `TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA`

However, if you want to enable any of the preceding anonymous ciphers, include the following argument in the Java command that starts WebLogic Server:

```
-Dweblogic.security.SSL.AllowAnonymousCipher=true
```

In most cases, enabling anonymous ciphers is required when WebLogic Server, or its deployed application, acts as a SSL client that is making an outbound connection to an SSL server (for example, an LDAP server or RDBMS system) that is configured to use anonymous ciphers only. A typical use case is connecting to an Oracle Internet Directory instance that is configured in no-auth mode.

> **Note:**
>
> Oracle does not recommend the use of anonymous ciphers in production environments.

Cipher Suite Name Equivalents

By default, Certicom cipher suite names are converted to `SunJSSE` cipher suite names when WebLogic Server is configured to use the JSSE-based SSL implementation. Table 38-2 lists each cipher suite supported in the (removed) WebLogic Server Certicom SSL implementation

and its SunJSSE equivalent. The TLS_ name is the Certicom cipher suite name; the SSL_ name is the equivalent SunJSSE provider cipher suite name.

Table 38-2 Cipher Suite Name Equivalence

Certicom Cipher Suite	SunJSSE Equivalent Cipher Suite
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA	SSL_DHE_DSS_WITH_DES_CBC_SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA	SSL_DHE_RSA_WITH_DES_CBC_SHA
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA	SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_anon_EXPORT_WITH_RC4_40_MD5	SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
TLS_DH_anon_WITH_DES_CBC_SHA	SSL_DH_anon_WITH_DES_CBC_SHA
TLS_DH_anon_WITH_RC4_128_MD5	SSL_DH_anon_WITH_RC4_128_MD5
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA	SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5	SSL_RSA_EXPORT_WITH_RC4_40_MD5
TLS_RSA_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_RC4_128_MD5	SSL_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA

Setting Cipher Suites Using WLST: An Example

The following example shows a WLST script that sets the cipher suites SSL_RSA_WITH_RC4_128_MD5, SSL_RSA_WITH_RC4_128_SHA, and SSL_RSA_WITH_3DES_EDE_CBC_SHA. After this script is run, the cipher suites are set in the domain configuration (that is, the config.xml file) and the SSL listeners are restarted with the new cipher suite settings.

> **Note:**
>
> For clarity, this WLST example script shows the username and password in clear text. However, you should avoid entering clear-text passwords in WLST commands in general, and you should especially avoid saving on disk WLST scripts that include clear-text passwords. In these instances you should use a mechanism for passing encrypted passwords instead. See Security for WLST in *Understanding the WebLogic Scripting Tool*.

```
url="t3://localhost:7001"
adminUsername="weblogic"
adminPassword="password"
connect(adminUsername, adminPassword, url)
edit()
```

```
server=cmo.lookupServer('myserver')
cd('Servers')
cd('myserver')
startEdit()
cd('SSL')
cd('myserver')
ssl = server.getSSL()
ciphers = ['SSL_RSA_WITH_RC4_128_MD5', 'SSL_RSA_WITH_RC4_128_SHA',
'SSL_RSA_WITH_3DES_EDE_CBC_SHA']
ssl.setCiphersuites(ciphers)
save()
activate()
disconnect()
exit()
```

An Important Note Regarding Null Cipher Use in SSL

A cipher suite is an SSL encryption method that includes the key exchange algorithm, the symmetric encryption algorithm, and the secure hash algorithm. A cipher suite is used to protect the integrity of a communication. For example, the cipher suite called RSA_WITH_RC4_128_MD5 uses RSA for key exchange, RC4 with a 128-bit key for bulk encryption, and MD5 for message digest.
SSL clients start the SSL handshake by connecting to the server. As part of the connection, the client sends the server a list of the cipher suites it supports. The server then selects a mutually-supported cipher suite from the list supplied by the client for the client and server to use for this session.

However, an incorrectly configured client might specify a set of cipher suites that contain only null ciphers. A null cipher passes data on the wire in clear-text. (An example of a cipher suite with a null cipher is SSL_RSA_WITH_NULL_MD5.) Using a null cipher makes it possible to see the SSL messages by using a network packet sniffer. In essence, SSL is used but does not provide any security.

The server selects the null cipher only when it is the only cipher suite they have in common. If the server selects a null cipher from the client's cipher suite list, the log contains the following message: SSL has established a session that uses a Null cipher.

This message is output only when the server has selected a null cipher from the client's list.

> **Note:**
>
> If there is any potential whatsoever that an SSL client might use a null cipher to inappropriately connect to the server, you should check the log file for this message. It is recommended that new client configurations be given extra attention with respect to the use of a null cipher to ensure that they are properly configured.
>
> It is unlikely that an existing client configuration would suddenly start using null ciphers if it had not been doing so previously. However, an existing client configuration that is unknowingly configured incorrectly could be using null ciphers.

Other SSL errors unrelated to null ciphers are possible as well, and each will display an appropriate error message in the log.

See Configuring SSL. For information on viewing log files, see View Logs in *Oracle WebLogic Remote Console Online Help*.

WebLogic Server Control to Prevent Null Cipher Use

WebLogic Remote Console includes a control to prevent the server from using a null cipher.

In the **Edit Tree**, go to **Environment**, then **Servers**, then *myServer*. On the **Security** tab, select the **SSL** subtab and enable **Show Advanced Fields**.

The **Allow Unencrypted Null Cipher** control determines whether null ciphers are allowed. By default, this control is disabled and the use of a null cipher is not allowed on the server. In such a configuration, if the SSL/TLS clients want to use the null cipher suite (by indicating SSL_RSA_WITH_NULL_MD5 as the only supported cipher suite), the SSL/TLS handshake will fail.

If you enable this control, the null cipher suite (for example, SSL_RSA_WITH_NULL_MD5) is added to the list of supported cipher suites by the server. The SSL/TLS connection has a chance to use the null cipher suite if the client wants to do so. If the null cipher suite is used, the message will be unencrypted.

> **Caution:**
>
> Do not set this control in a production environment unless you are aware of the implications and consequences of doing so.

This control is also exposed as a system runtime parameter, `weblogic.security.SSL.allowUnencryptedNullCipher`, and as an `AllowUnencryptedNullCipher` attribute on the SSLMBean.

> **Note:**
>
> TLS anon and NULL cipher suites are disabled by default in the JDK.

Using Debugging with JSSE SSL

JSSE SSL debugging provides detailed information about the SSL events that occurred during an SSL handshake and other operations.See SSL Debugging.
If you debug SSL when the JSSE-based SSL implementation is enabled, you can use the logging properties listed and described in Table 38-1. However, some properties affect only the SSL calling code and not the JSSE implementation. The JSSE-based SSL implementation has its own logging system, which is activated by the `javax.net.debug` property. The `javax.net.debug` property provides multiple levels of control over the amount of output and can be used independently of WebLogic SSL logging (`ssl.debug`).

For more information about the `javax.net.debug` property, see Debugging Utilities - Java Secure Socket Extension (JSSE) Reference Guide in *Security Developer's Guide*.

39

X.509 Certificate Revocation Checking

Learn about the X.509 certificate revocation (CR) checking feature, which is supported in Oracle WebLogic Server's JSSE implementation. This feature checks a certificate's revocation status as part of the SSL certificate path validation process. CR checking improves the security of certificate usage by ensuring that received certificates have not been revoked by the issuing certificate authority.

Certificate Revocation Checking Overview

In WebLogic Server, Certificate Revocation (CR) checking can be used for several purposes including, validating client certificates (inbound SSL) and server certificates (outbound SSL).

WebLogic Server's CR checking mechanism includes the following features:

- Support for the following certificate revocation methods:
  - Online Certificate Status Protocol (OCSP)
  - Certificate revocation lists (CRLs)
- You can configure CR checking on a domain-wide basis for all certificate authorities (CAs). And optionally, you can also configure certificate authority overrides for specific CAs.

  A certificate authority override contains changes to the domain-wide CR checking configuration that you want to have in effect for certificates that have been issued by a specific CA. For example, you can configure a particular OCSP responder URL to be used, or require SSL certificate path validation to fail if certificate revocation status cannot be determined. Each certificate authority override you create applies to only one specific CA.

CR checking is disabled by default in WebLogic Server. But using either WebLogic Remote Console or WLST, you can enable CR checking and configure the properties described in the sections that follow.

> **Note:**
>
> CR checking is available for a WebLogic Server instance only when JSSE is enabled.

Enabling the Default CR Checking Configuration

In WebLogic Server, CR checking is disabled by default. When you enable CR checking, WebLogic Server provides, on a domain-wide basis, a comprehensive set of mechanisms to obtain current revocation status of each certificates it validates.

This topic describes the default behavior WebLogic Server provides when you enable CR checking. The subsequent sections explain customizations you can make that can be applied domain-wide or, selectively, to specific certificate authorities.

When the default CR checking configuration is enabled, WebLogic Server automatically does the following when performing SSL certificate path validation:

1. Checks the **OCSP response local cache** to obtain certificate revocation status. The OCSP response local cache is an in-memory cache that holds the latest certificate status that is provided by **OCSP responders**.

   Certificate status in OCSP has a specific validity period. If the certificate status has expired, WebLogic Server does the following:

   a. Obtains the **OCSP responder URI** from the certificate. This URI is included in the Authority Information Access (AIA) value in the certificate, which indicates how to access information and services from the issuer of the certificate.

   b. Submits an **OCSP request** to the OCSP responder.

      The OCSP responder returns an **OCSP response**, which includes a certificate status of `good`, `revoked`, or `unknown`.

   c. Updates the OCSP response local cache with the OCSP response.

   For certificates that have a valid, non-expired entry in the OCSP response local cache, WebLogic Server can obtain its revocation status from the cache instead of requesting a fresh OCSP response. This provides improved performance and reduced use of network bandwidth.

   > **Note:**
   >
   > Note the following:
   >
   > - Cached entries expire based on the OCSP validity period, but the cache behavior can be customized.
   > - The local OCSP response cache is never used when OCSP nonce is enabled. This ensures the freshest response.

2. If the certificate has an OCSP status of `unknown`, WebLogic Server checks the **CRL local cache** for valid CRLs to determine whether the certificate has been revoked. (If either a `revoked` or `not revoked` status is determined by OCSP, CRL is not used for the certificate.)

   By default, the CRL local cache is a file-based store that is maintained on each server instance in a WebLogic domain and that is updated on demand from **CRL distribution points**. A CRL distribution point is a network-accessible server that provides CRLs for download.

   If no valid CRLs are available in the CRL local cache, WebLogic Server does the following:

   a. Obtains the **CRL distribution point URL**, which is included in the CRLDistributionPoints extension in the certificate.

b. Using the CRL distribution point URL, downloads a fresh CRL and adds it to the cache.

c. Searches the CRL for an entry that corresponds to the certificate.

If the certificate serial number is not found in the CRL from the issuer, the certificate status is set to `not revoked`.

Note the following:

- If the certificate has an OCSP status of `revoked`, or is included in a valid CRL, WebLogic Server automatically fails SSL certificate path validation.
- If the revocation status is unknown or cannot be determined after using OCSP and checking the available CRLs, certificate path validation by default is *not* failed.

The following topics explain how to configure and customize default CR checking:

- Configuring Default CR Checking
- Customizing the CR Checking Configuration

Configuring Default CR Checking

Enabling the default CR checking capability in a WebLogic domain is available through the following MBean attribute:

Table 39-1 MBean Attributes

MBean Attribute	Description	Default Value
`CertRevocMBean.CheckingEnabled`	Specifies whether CR checking is enabled domain-wide.	`False`

For information about how to use WebLogic Remote Console to enable CR checking in a WebLogic domain, see Enable Certificate Revocation Checking in *Oracle WebLogic Remote Console Online Help*.

You can configure a CA override for this MBean attribute, as explained in Configuring Certificate Authority Overrides.

Customizing the CR Checking Configuration

The default CR checking behavior in WebLogic Server is appropriate for deployment environments in which CR checking is desired, but not required. Depending on your environment, you might require CR checking, or need to enforce behaviors that are specific to particular certificate authorities. Table 39-2 lists and summarizes the types of customizations you can make to CR checking in WebLogic Server and provides links to the sections in which they are explained.

Table 39-2 Customizations You Can Make to the CR Checking Configuration

Customization	Description
CR checking method order	Specifies the order in which the supported CR checking methods are used; that is, OCSP and CRLs. Optionally, you can choose to use only OCSP, or only CRLs. See Choosing the CR Checking Methods to Be Used by WebLogic Server.

Table 39-2 (Cont.) Customizations You Can Make to the CR Checking Configuration

Customization	Description
Require certificate revocation status	Specifies that SSL certificate path validation must fail if a certificate's revocation status is unknown or cannot be determined. See Failing SSL Certificate Path Validation if Revocation Status Cannot Be Determined .
Domain-wide OCSP settings	Customize, domain-wide, one or more of the following OCSP features or behaviors: • Use of nonces in OCSP requests and responses • OCSP response cache. For example, capacity or refresh period • OCSP response timeout interval settings See Using the Online Certificate Status Protocol.
Domain-wide CRL protocol settings	Customize, domain-wide, one or more of the following CRL features or behaviors: • Use of CRL distribution points • CRL cache refresh frequency • CRL distribution point download timeout interval settings See Using Certificate Revocation Lists.
Certificate authority overrides	Customize the CR checking behavior for certificates issued by a particular CA. For example: • Disable revocation checking for those certificates • Change the CR checking method order • Automatically fail certificate path validation if revocation status is unknown or unavailable • Customize OCSP or CRL settings (except for the CRL local cache settings) • Designate the OCSP responder URL to use • Designate the CRL distribution point URL to use A certificate authority override always takes precedence over domain-wide settings that are in place. See Configuring Certificate Authority Overrides.

Choosing the CR Checking Methods to Be Used by WebLogic Server

By default, when checking a certificate's revocation status, WebLogic Server first uses Online Certificate Status Protocol (OCSP). If OCSP returns the certificate's status as "unknown," WebLogic Server then uses CRLs. However, you can change the CR checking method and order in a WebLogic domain by using the `CertRevocMBean.MethodOrder` MBean attribute.

You can change the CR checking method used, or the sequence in which the methods are used, to one of the following:

- OCSP only
- CRLs only
- OCSP then CRLs — If the OCSP status for a certificate is returned as `unknown`, CRLs are checked for certificate status.
- CRLs then OCSP — If a certificate's revocation status cannot be determined by checking available CRLs, its OCSP status is checked.

Configuring the CR checking method and order in a WebLogic domain is available through the following MBean attribute:

Table 39-3 MBean Attributes

MBean Attribute	Description	Default Value
`CertRevocMBean.MethodOrder`	Specifies the domain-wide CR checking method.	`OCSP_THEN_CRL`

You can configure a CA override for this MBean attribute, as explained in Configuring Certificate Authority Overrides.

For information about how to use WebLogic Remote Console to configure the CR checking method and order for a WebLogic domain, see Enable Certificate Revocation Checking in *Oracle WebLogic Remote Console Online Help*.

Failing SSL Certificate Path Validation if Revocation Status Cannot Be Determined

By default, if an X.509 certificate's revocation status cannot be determined by any of the selected checking methods, the certificate can still be accepted if the SSL certificate path validation is otherwise successful. However, for certificates whose revocation status cannot be determined, you can optionally configure WebLogic Server to fail certificate path validation.

Configuring a WebLogic domain to fail SSL certificate path validation when the revocation status cannot be determined is available through the following MBean attribute:

Table 39-4 MBean Attributes

MBean Attribute	Description	Default Value
`CertRevocMBean.FailOnUnknownRevocStatus`	Specifies on a domain-wide basis whether a certificate's path validation should fail if its revocation status cannot be determined.	`False`

You can configure a CA override for this MBean attribute, as explained in Configuring Certificate Authority Overrides.

For information about how to configure this MBean attribute using WebLogic Remote Console, see Enable Certificate Revocation Checking in *Oracle WebLogic Remote Console Online Help*.

Using the Online Certificate Status Protocol

The Online Certificate Status Protocol (OCSP) is an automated certificate checking network protocol that is defined in RFC 2560.

As part of certificate validation, WebLogic Server queries the revocation status of a certificate by issuing an **OCSP request** to an **OCSP responder**. Certificate status is maintained by the OCSP responder. Acceptance of the certificate is suspended until the responder returns an **OCSP response**, indicating whether the certificate is still trusted by the CA that issued it.

OCSP may be used to satisfy some of the operational requirements of providing more timely revocation information than is possible with CRLs and may also be used to obtain additional

status information. For more information about OCSP, see the description of RFC 2560 at http://www.ietf.org/rfc/rfc2560.txt.

The following sections describe how to configure OCSP in WebLogic Server:

- Using Nonces in OCSP Requests
- Setting the Response Timeout Interval
- Enabling and Configuring the OCSP Response Local Cache

Using Nonces in OCSP Requests

A nonce is a random number that, when included in an OCSP request, forces a fresh response; pre-signed responses are rejected. The use of nonces can prevent replay attacks. By default, WebLogic Server does *not* include nonces in OCSP requests.

However, when WebLogic Server is configured to use nonces in OCSP:

1. WebLogic Server generates a nonce for each OCSP request, and includes it in an extension in the request.
2. The signed OCSP response must include the same nonce, which is included in an extension in the response.

You can configure the use of OCSP nonces in a WebLogic domain using the following MBean attribute:

Table 39-5 MBean Attributes

MBean Attribute	Description	Default Value
`CertRevocMBean.OcspNonceEnabled`	Specifies whether nonces are generated for OCSP requests. This setting is domain-wide.	`false`

You can also configure CA overrides for this MBean attribute. See Configuring OCSP Properties in a Certificate Authority Override.

For information about how to use WebLogic Remote Console to configure OCSP nonces, see Enable Certificate Revocation Checking in *Oracle WebLogic Remote Console Online Help*.

Setting the Response Timeout Interval

The response timeout interval limits the wait time for OCSP responses. Setting a timeout interval helps minimize blocked threads and also reduces the system's vulnerability to denial of service attacks. In addition to setting a response timeout interval, you can configure a time tolerance value for handling clock-skew differences between WebLogic Server and OCSP responders.

The default response timeout interval is 10 seconds, with a zero time tolerance. The response timeout interval and time tolerance value can be set domain-wide and, optionally, set specific to one or more CAs.

You can configure the OCSP response timeout interval and time tolerance value for a WebLogic domain using the following MBean attributes:

Table 39-6 MBean Attributes

MBean Attribute	Description	Default Value
`CertRevocMBean.OcspResponseTimeout`	Specifies the domain-wide timeout interval, in seconds, for OCSP responses. The valid range is between 1 and 300, inclusive.	`10`
`CertRevocMBean.OcspTimeTolerance`	Specifies the domain-wide OCSP time tolerance value, in seconds, for OCSP responses.	`0`

You can also configure CA overrides for these MBean attributes. See Configuring OCSP Properties in a Certificate Authority Override.

For information about how to use WebLogic Remote Console to configure OCSP response timeout interval and time tolerance values, see Enable Certificate Revocation Checking in *Oracle WebLogic Remote Console Online Help*.

Enabling and Configuring the OCSP Response Local Cache

To optimize performance and reduce network bandwidth, WebLogic Server's OCSP implementation is configured by default to use a local in-memory cache for holding OCSP responses, called the **OCSP response local cache**. Cached entries automatically expire based on the OCSP validity period and other criteria, such as entries least accessed. If nonces are enabled, OCSP responses obtained using a nonce are not cached. This ensures the freshest response is always used with nonces.

You can configure the OCSP response local cache in a WebLogic domain using the following MBean attributes:

Table 39-7 MBean Attributes

MBean Attribute	Description	Default Value
`CertRevocMBean.OcspResponseCacheEnabled`	Specifies whether the OCSP response local cache is enabled domain-wide.	`true`
`CertRevocMBean.OcspResponseCacheCapacity`	Specifies the maximum number of entries supported by the OCSP response local cache.	`1024`
`CertRevocMBean.OcspResponseCacheRefreshPeriodPercent`	Specifies the refresh period for the OCSP response local cache, expressed as a percentage of the validity period of the response. For example, for a validity period of 10 hours, a value of 10% specifies that after one hour, the cached response expires and a fresh response is required.	`100`

You can also configure CA overrides for this MBean attribute.See Configuring OCSP Properties in a Certificate Authority Override.

For information about how to use WebLogic Remote Console to configure the OCSP response local cache, see Enable Certificate Revocation Checking in *Oracle WebLogic Remote Console Online Help*.

Using Certificate Revocation Lists

A certificate revocation list (CRL) is a time-stamped list of digital certificates that have been revoked by the certificate authority (CA) that issued them. Each CRL is signed by a CA and is made available in a public repository. The WebLogic Server CRL implementation provides a CRL local cache for more efficient CR checking, automatic import of user CRL files, and distribution points from which the cache can be populated and refreshed.

The CRL implementation in WebLogic Server includes support for the following:

- CRL local cache, which enables efficient access for CR checking.
- Automatic import of user supplied CRL files into the CRL cache.
- Use of distribution points from which the CRL cache can optionally be populated and refreshed.

The following sections explain how to configure CRL usage in WebLogic Server:

- Enabling Updates from Distribution Points
- Configuring the CRL Local Cache

Enabling Updates from Distribution Points

Updating CRLs from distribution points is enabled by default. If the appropriate CRL for a certificate being validated does not already exist in the local cache, the CRL is downloaded from an available distribution point.

WebLogic Server also allows you to configure a timeout interval for the CRL download from a distribution point. This timeout interval limits the wait time for CRL downloads, and also minimizes the risk of blocked threads and vulnerability to denial of service attacks. Note that if the CRL download times out, the CRL method reports that the revocation status is unknown; however, the CRL download continues in a separate thread until complete and the CRL becomes available for future CRL checking.

You can configure CRL distribution points for a WebLogic domain using the following MBean attributes:

Table 39-8 MBean Attributes

MBean Attribute	Description	Default Value
`CertRevocMBean.CrlDpEnabled`	Specifies whether CRL distribution points are enabled domain-wide.	`true`
`CertRevocMBean.CrlDpDownloadTimeout`	Specifies the overall timeout interval, domain-wide, for the distribution point CRL download, expressed in seconds. The valid range is between 1 and 300, inclusive.	`10`

You can also configure CA overrides for these MBean attributes. See Configuring CRL Properties in a Certificate Authority Override.

For information about how to use WebLogic Remote Console to configure CRL distribution points for a WebLogic domain, see Enable Certificate Revocation Checking in *Oracle WebLogic Remote Console Online Help*.

Configuring the CRL Local Cache

The CRL local cache is automatically enabled in WebLogic Server. Because obtaining CRLs is a time-consuming process, CRLs can be stored, while valid, in local files. In addition, WebLogic Server allows you to configure the refresh interval for the local cache, expressed as a percentage of the validity period of the CRL.

You may supply CRL files to be used by copying them into the following CRL import directory, where *server-name* represents the name of the WebLogic Server instance:

WL_HOME/servers/*server-name*/security/certrevocation/crlcache/import

The CRL files are automatically imported and internally cached. This directory is automatically created, if it does not already exist, when CR checking is enabled and an SSL connection is attempted.

> **Note:**
>
> Note the following:
>
> - After WebLogic Server is started, the import of the CRL file starts automatically when CR checking is enabled and at least one attempt to check a certificate's revocation status has occurred. This minimizes resource usage until necessary.
> - After you import CRL files, they are automatically deleted from the import directory.
> - The CRL local cache configuration settings are domain-wide. You cannot configure a certificate authority override for the CRL local cache.

You can configure the CRL local cache for a WebLogic domain using the following MBean attributes:

Table 39-9 MBean Attributes

MBean Attribute	Description	Default Value
`CertRevocMBean.CrlCacheRefreshPeriodPercent`	Specifies the refresh period for the CRL local cache, expressed as a percentage of the validity period of the CRL.	`100`

For information about how to use WebLogic Remote Console to configure the CRL local cache for a WebLogic domain, see Enable Certificate Revocation Checking in *Oracle WebLogic Remote Console Online Help*.

Configuring Certificate Authority Overrides

Configuring certificate authority overrides allows you to specify CR checking behavior that is enforced for certificates issued by a particular CA. A certificate authority override always supersedes the domain-wide CR checking configuration that is enabled.

The following sections explain how to configure CR checking CA overrides:

- General Certificate Authority Overrides

- Configuring OCSP Properties in a Certificate Authority Override
- Configuring CRL Properties in a Certificate Authority Override

General Certificate Authority Overrides

To create a certificate authority override for a specific CA, complete the following steps:

1. Identify the CA by its distinguished name. This must be the complete issuer distinguished name (defined in RFC 2253) of the certificates for which this override applies.

   For example, the distinguished name of the WebLogic Server DemoTrust CA is `CN=CertGenCA_domain_name, OU=FOR TESTING ONLY, O=MyOrganization, L=MyTown, ST=MyState, C=US`.
2. Specify whether CR checking is enabled for certificates issued by this CA, if necessary.
3. Specify the CR checking methods and order performed for certificates issued by this CA.
4. Specify whether SSL certificate path validation should fail if the revocation status of certificates issued by this CA cannot be determined.
5. Optionally, specify additional OCSP or CRL customizations, as explained in the following sections:
   - Configuring OCSP Properties in a Certificate Authority Override
   - Configuring CRL Properties in a Certificate Authority Override

You can configure general certificate authority overrides for a CA by using the following MBean attributes:

Table 39-10 MBean Attributes

MBean Attribute	Description	Default Value
`CertRevocCaMBean.DistinguishedName`	Specifies the distinguished name (DN) of the CA subject.	None (required field)
`CertRevocCaMBean.CheckingDisabled`	For this CA, specifies whether CR checking is disabled.	`false`
`CertRevocCaMBean.FailOnUnknownRevocStatus`	For this CA, specifies whether SSL certificate path checking should fail if the certificate revocation status cannot be determined from any of the available methods.	Same as current setting of `CertRevocMBean.FailOnUnknownRevocStatus`.
`CertRevocCaMBean.MethodOrder`	Specifies the certificate revocation checking method order when checking certificates issued by this CA.	Same as current setting of `CertRevocMBean.MethodOrder`.

For information about how to use WebLogic Remote Console to configure certificate authority overrides, see Configure Certificate Authority Overrides in *Oracle WebLogic Remote Console Online Help*.

Configuring OCSP Properties in a Certificate Authority Override

WebLogic Server tries the following trust models in its OCSP implementation:

- Delegated Trust Model (DTM) — The OCSP response is signed by an OCSP responder that has been delegated by the CA to sign responses on its behalf.
- Explicit Trust Model (ETM) — If neither the CA nor an authority to which OCSP responsibilities have been delegated has signed the OCSP response, an explicitly trusted signer may be specified. ETM is used when you can supply an additional trusted certificate that may be used to verify the OCSP response signature. This can be any certificate, including one unrelated to the CA corresponding to the override. ETM may be used for OCSP responders which are trusted, but are not authorized to sign OCSP responses on behalf of issuers. Explicitly trusted public certificates for OCSP responders may be suitable if the OCSP server is internally maintained within your enterprise.
- CA-signed Trust Model — The OCSP response is presumed to be signed by the same CA that issued the certificate for which the revocation status is being requested.

When you create a certificate authority override, WebLogic Server allows you to configure the OCSP properties that are described in Table 39-11. This table also identifies the MBean attributes you can use to configure these override properties.

Table 39-11 OCSP Properties That Can Be Specified in a Certificate Authority Override

Override	Description	MBean Attribute
OCSP responder URL	Specifies the URL to be used for either: • Failover, if the OCSP responder URI from the certificate AIA value is not available or not acceptable • Override, to be always used as the responder URL instead of the responder URI from the certificate AIA. See Identifying the OCSP Responder URL.	`CertRevocCaMBean.OcspResponderUrl` The default value is none.
How the OCSP responder URL is used	Specifies how the OCSP responder URL is to be used: for failover or override.	`CertRevocCaMBean.OcspResponderUrlUsage` The default value is `FAILOVER`.
OCSP responder certificate subject name	For this CA, specifies the explicitly trusted OCSP responder certificate subject name. For example, `CN=OCSP Responder, O=XYZ Corp`. This must correspond to the subject distinguished name of a certificate in the configured WebLogic Server trust keystore. In cases where the subject name alone is not sufficient to uniquely identify the certificate, both the `CertRevocCaMBean.OcspResponderCertIssuerName` and `CertRevocCaMBean.OcspResponderCertSerialNumber` are used instead.	`CertRevocCaMBean.OcspResponderCertSubjectName` The default value is `NONE`.

Table 39-11 (Cont.) OCSP Properties That Can Be Specified in a Certificate Authority Override

Override	Description	MBean Attribute
OCSP responder certificate issuer name	For this CA, specifies the explicitly trusted OCSP responder certificate issuer name. For example, `CN=Enterprise CA, O=XYZ Corp`. This must correspond to the issuer distinguished name of a certificate in the configured WebLogic Server trust keystore. When this attribute is set, the `CertRevocCaMBean.OcspResponderCertSerialNumber` must also be set.	`CertRevocCaMBean.OcspResponderCertIssuerName` The default value is `NONE`.
OCSP responder certificate serial number	For this CA, specifies the explicitly trusted OCSP responder certificate serial number. For example, `2A:FF:00`. This must correspond to the serial number of a certificate in the configured WebLogic Server trust keystore. When this attribute is set, the `CertRevocCaMBean.OcspResponderCertIssuerName` attribute must also be set.	`CertRevocCaMBean.OcspResponderCertSerialNumber` The default value is `NONE`.
OCSP responder Explicit Trust Method	For this CA, specifies whether the OCSP Explicit Trust model is enabled and how a trusted certificate in the Weblogic Server trust keystore is specified. The following values can be specified: • `NONE` specifies that Explicit Trust is disabled. • `USE_SUBJECT` specifies that the trusted certificate is identified using the subject DN that is specified in the `CertRevocCaMBean.OcspResponderCertSubjectName` attribute. • `USE_ISSUER_SERIAL_NUMBER` specifies that the trusted certificate is identified using the issuer DN and certificate serial number that are specified in the `CertRevocCaMBean.OcspResponderCertIssuerName` and `CertRevocCaMBean.OcspResponderCertSerialNumber` attributes, respectively.	`CertRevocCaMBean.OcspResponderExplicitTrustMethod` The default value is `NONE`.
Nonce enabled	For this CA, specifies whether nonces are sent with OCSP requests, which forces a fresh (not pre-signed) response.	`CertRevocCaMBean.OcspNonceEnabled` The default value is the same as the current setting for `CertRevocMBean.OcspNonceEnabled`.

Table 39-11 (Cont.) OCSP Properties That Can Be Specified in a Certificate Authority Override

Override	Description	MBean Attribute
OCSP response local cache	For this CA, specifies whether the OCSP response local cache is enabled.	`CertRevocCaMBean.OcspResponseCacheEnabled` The default value is the same as the current setting for `CertRevocMBean.OcspResponseCacheEnabled`.
OCSP response timeout	For this CA, specifies the timeout interval for the OCSP response, expressed in seconds. The valid range is between 1 and 300, inclusive. See Setting the Response Timeout Interval.	`CertRevocCaMBean.OcspResponseTimeout` The default value is the same as the current setting for `CertRevocMBean.OcspResponseTimeout`.
OCSP time tolerance	For this CA, specifies the time tolerance value for handling clock-skew differences between WebLogic Server and responders, expressed in seconds. The valid range is between 0 and 900, inclusive. The validity period of the response is extended both into the future and into the past by the specified amount of time, effectively widening the validity interval.	`CertRevocCaMBean.OcspTimeTolerance` The default value is the same as the current setting for `CertRevocMBean.OcspTimeTolerance`.

For information about how to use WebLogic Remote Console to configure OCSP settings in a certificate authority override, see Configure Certificate Authority Overrides in *Oracle WebLogic Remote Console Online Help*.

The following topic explains how to identify the OCSP Responder URL:

Identifying the OCSP Responder URL

To validate a certificate using an OCSP responder lookup, WebLogic Server uses the following methods to determine the OCSP responder URL:

- Authority Information Access (AIA) value in the certificate, which indicates how to access information and services for the issuer of the certificate. For example, the AIA contains the URI for the OCSP responder.
- Default OCSP responder failover or override — If the OCSP responder URI is not available from the certificate AIA value, or is not acceptable, a default OCSP responder URL can be configured on a per-CA basis.

  Additionally, the default OCSP responder URL per CA can be specified selectively for either failover, or for override. When specified for override, this URL always overrides the value obtained from the certificate AIA extension.

For information about how to use WebLogic Remote Console to set the OCSP responder URL in a certificate authority override, see Configure Certificate Authority Overrides in *Oracle WebLogic Remote Console Online Help*.

Configuring CRL Properties in a Certificate Authority Override

When you configure a certificate authority override, WebLogic Server allows you to configure the CRL properties listed and described in Table 39-12. This table also identifies the MBean attributes you can use to configure these properties.

Table 39-12 CRL Properties That Can Be Specified in a Certificate Authority Override

Override	Description	MBean Attribute
Use of distribution point to update local CRL cache	For this CA, specifies whether CRL distribution point processing to update the local CRL cache is enabled.	`CertRevocCaMBean.CrlDpEnabled` The default value is the same as the current setting for `CertRevocMBean.CrlDpEnabled`.
Distribution point URL	For this CA, specifies the CRL distribution point URL to be used for either: • Failover, if the URL from the CRLDistributionPoints extension in the certificate is unavailable • Override, to be always used as the CRL distribution point URL instead of the CRLDistributionPoints extension in the certificate	`CertRevocCaMBean.CrlDpUrl` The default value is `null`.
How the distribution point URL is used	Specifies how the distribution point URL is to be used: for failover or override.	`CertRevocCaMBean.CrlDpUrlUsage` The default value is `FAILOVER`.
Distribution point CRL download timeout	For this CA, specifies the overall timeout interval for the distribution point CRL download, expressed in seconds. The valid range is between 1 and 300, inclusive.	`CertRevocCaMBean.CrlDpDownloadTimeout` The default value is the same as the current setting for `CertRevocMBean.CrlDpDownloadTimeout`.

For information about how to use WebLogic Remote Console to customize the CRL settings in a certificate authority override, see Configure Certificate Authority Overrides in *Oracle WebLogic Remote Console Online Help*.

40

Configuring an Identity Keystore Specific to a Network Channel

Learn how to configure a network channel to have its own custom identity keystore, and other SSL attributes, that are separate from and that override the default keystore and SSL configuration settings for the Managed Server instance or the domain. This feature enables you to configure an Oracle WebLogic Server instance to use one identity and SSL configuration on one network channel, and another identity and SSL configuration on other channels.

About Network Channels

A network channel in a WebLogic Server instance is a combination of the four attributes — communication protocol (which can be t3, t3s, http, or https), listen address, listen port, and channel name.

See Understanding Network Channels in *Administering Server Environments for Oracle WebLogic Server*,

By default, when you configure a network channel, the channel uses the SSL configuration that is set for the server instance. This means that the channel uses the same identity and trust that is established for the server. The server might use a custom identity that is specific to that server, or it might be a single domain-wide identity, depending on how the server instance and domain are configured.

However, rather than using one identity for all network communication in which a Managed Server instance participates, you might have a need for the server to switch to a different identity when communicating with a particular client. For example, you might need to use one identity for the server when communicating with one particular business group, and a different identity for the server when communicating with other Managed Server instances in the domain. By customizing a network channel to use a custom identity keystore that is separate from either the identity keystore configured for the server instance or the one configured for the domain, you can assert one identity on one network channel, and another identity on a different channel.

Channel-Specific SSL Configuration Attributes

The `NetworkAccessPointMBean` contains the attributes that you can set to create a channel-specific SSL configuration. In addition to enabling a network channel to use a custom identity keystore, these attributes also allow you to customize other SSL settings, such as the use of a custom host name verifier, the cipher suites to be used in SSL communications, and certificate validation rules.

Table 40-1 lists and describes the SSL attributes that can be configured on the `NetworkAccessPointMBean` for a specific network channel.

> **Note:**
>
> For ease of reference in Table 40-1, the following attributes on the `NetworkAccessPointMBean` are referred to collectively as the `CustomIdentityKeyStore*` attributes:
>
> - `CustomIdentityKeyStoreFileName`
> - `CustomIdentityKeyStorePassPhrase`
> - `CustomIdentityKeyStorePassPhraseEncrypted`
> - `CustomIdentityKeyStoreType`

Table 40-1 NetworkAccessPointMBean Attributes for Customizing a Channel's SSL Configuration

Attribute	Description
ChannelIdentityCustomized	Specifies whether the channel's custom identity should be used. This setting has an effect only if the network channel uses a custom keystore. By default the channel's identity is inherited from the server's identity. The `CustomIdentityKeyStore*` attributes have the following validation rules related to the `ChannelIdentityCustomized` attribute to ensure that the network channel alias relates to the channel keystore and does not default to an alias in the server keystore: 1. If *any* `CustomIdentityKeyStore*` attributes are set, then *all* `CustomIdentityKeyStore*` attributes must be set. 2. The `ChannelIdentityCustomized` attribute must be set to `true`. 3. The `CustomPrivateKeyAlias` attribute must be set. Note that if the `CustomIdentityKeyStore*` attributes are *not* set, the `CustomPrivateKeyAlias` attribute may be set to refer to the server keystore.
CustomIdentityKeyStoreFileName	Specifies the custom identity keystore to assign to the channel. If a value for this attribute is not set, the value of the `ServerMBean.CustomIdentityKeyStoreFileName` attribute is used by default. This attribute is used only if the `ServerMBean.KeyStores` attribute is set to one of the following values: • `CUSTOM_IDENTITY_AND_JAVA_STANDARD_TRUST` • `CUSTOM_IDENTITY_AND_CUSTOM_TRUST` • `CUSTOM_IDENTITY_AND_COMMAND_LINE_TRUST` If you are using a JKS or PKCS12 keystore, specify this value as an absolute path, or as a relative path to the directory from which the server is booted. See Configuring Keystores. If you are using an Oracle OPSS Key Store Service (KSS) keystore, specify this value as the KSS URI. See Configuring the OPSS Keystore Service for Custom Identity and Trust: Main Steps.

Table 40-1 (Cont.) NetworkAccessPointMBean Attributes for Customizing a Channel's SSL Configuration

Attribute	Description
CustomIdentityKeyStorePassPhrase	Encrypts and decrypts the plain text form of the passphrase for the channel's custom identity keystore. When you set the keystore password using this attribute, WebLogic Server automatically encrypts the value and stores it in the `CustomIdentityKeyStorePassPhraseEncrypted` attribute. If the value is empty or null, keystores not requiring a passphrase may be opened. If a value for this attribute is not set, the value of the `ServerMBean.CustomIdentityKeyStorePassPhrase` attribute is used by default. This attribute is used only if the `ServerMBean.KeyStores` attribute is set to one of the following values: • `CUSTOM_IDENTITY_AND_JAVA_STANDARD_TRUST` • `CUSTOM_IDENTITY_AND_CUSTOM_TRUST` • `CUSTOM_IDENTITY_AND_COMMAND_LINE_TRUST` **Note:** Using the `CustomIdentityKeyStorePassPhrase` attribute is a potential security risk because the String object that contains the unencrypted password remains in the JVM memory until garbage collection removes it and the memory is reallocated, which potentially can be an indefinite duration. Therefore, Oracle recommends using the `CustomIdentityKeyStorePassPhraseEncrypted` attribute instead.
CustomIdentityKeyStorePassPhraseEncrypted	Specifies the encrypted passphrase that is set when the custom identity keystore is created. If a value for this attribute is not set, the value of the `ServerMBean.CustomIdentityKeyStorePassPhraseEncrypted` attribute is used by default. This attribute is only used if the `ServerMBean.KeyStores` attribute is set to one of the following values: • `CUSTOM_IDENTITY_AND_JAVA_STANDARD_TRUST` • `CUSTOM_IDENTITY_AND_CUSTOM_TRUST` • `CUSTOM_IDENTITY_AND_COMMAND_LINE_TRUST`
CustomIdentityKeyStoreType	Specifies the keystore type of the custom identity keystore. If you are using a JKS keystore, specify the value as `JKS`. If you are using a PKCS12 keystore, specify the value as `PKCS12`. If you are using the Oracle OPSS Key Store Service, specify this value as `KSS`. If a value for this attribute is not set, the value of the `ServerMBean.CustomIdentityKeyStoreType` attribute is used by default. The value of this attribute is used only if the `ServerMBean.KeyStores` attribute is set to one of the following values: • `CUSTOM_IDENTITY_AND_JAVA_STANDARD_TRUST` • `CUSTOM_IDENTITY_AND_CUSTOM_TRUST` • `CUSTOM_IDENTITY_AND_COMMAND_LINE_TRUST`
ClientCertificateEnforced	Specifies whether clients must present digital certificates from a trusted certificate authority to WebLogic Server on this channel.

Table 40-1 (Cont.) NetworkAccessPointMBean Attributes for Customizing a Channel's SSL Configuration

Attribute	Description
CustomPrivateKeyAlias	Specifies the string alias used to store and retrieve the channel's private key in the custom identity keystore. This private key is associated with the server's digital certificate. A value of `null` indicates that the network channel uses the alias specified in the server's SSL configuration. Note that if the `CustomIdentityKeyStore*` attributes are *not* set, the `CustomPrivateKeyAlias` attribute may be set to refer to the server keystore.
CustomPrivateKeyPassPhrase	Encrypts and decrypts the plain text form of the passphrase used to retrieve the channel's private key from the custom identity keystore. When you set the private key passphrase using this attribute, WebLogic Server automatically encrypts the value and stores it in the `CustomPrivateKeyPassPhraseEncrypted` attribute. This passphrase is assigned to the private key when it is generated. A value of `null` indicates that the network channel uses the passphrase specified in the server's SSL configuration.
CustomPrivateKeyPassPhrase Encrypted	Specifies the encrypted passphrase used to retrieve the channel's private key from the custom identity keystore.
OutboundPrivateKeyEnabled	Specifies whether the identity specified by the `NetworkAccessPointMBean.CustomPrivateKeyAlias` attribute should be used for outbound SSL connections on this channel. Typically the outbound identity is determined by the caller's environment.
TwoWaySSLEnabled	Specifies whether this network channel uses two way SSL.
HostnameVerificationIgnored	Specifies whether to ignore the configured implementation of the host name verifier (`weblogic.security.SSL.HostnameVerifier`). This attribute is used only when the server is acting as a client to another application server on a remote host. If a value for this attribute is not set, the value of the `SSLMBean.HostnameVerificationIgnored` attribute is used by default.
HostnameVerifier	Specifies the name of the class that implements the `weblogic.security.SSL.HostnameVerifier` interface. A host name verifier is useful when an SSL client (for example, WebLogic Server acting as an SSL client) connects to an application server on a remote host. The host name verifier helps to prevent man-in-the-middle attacks: It ensures that the host name in the URL to which the client connects matches the host name in the digital certificate that the server sends back as part of the SSL connection. If a value for this attribute is not set, the value of the `SSLMBean.HostnameVerifier` attribute is used by default.
Ciphersuites	Specifies the cipher suites that are to be used with the SSL listener for the network channel. During the SSL handshake, the strongest negotiated cipher suite is chosen. The cipher suites that are enabled by default depends on the specific JDK version with which WebLogic Server is configured. See Cipher Suites. If a value for this attribute is not set, the value of the `SSLMBean.Ciphersuites` attribute is used by default.

Table 40-1 (Cont.) NetworkAccessPointMBean Attributes for Customizing a Channel's SSL Configuration

Attribute	Description
AllowUnencryptedNullCipher	Specifies whether unencrypted null ciphers are allowed on the network channel. If a value for this attribute is not set, the value of the `SSLMBean.AllowUnencryptedNullCipher` attribute is used by default. During the SSL handshake, when the server and client negotiate the set of cipher suites that are to be used, the client might specify a set of cipher suites that contain only null ciphers. A null cipher passes data on the wire in clear-text, making it possible for a network packet sniffer to see the SSL messages. When null ciphers are used, SSL may be used for authentication, but messages may not be encrypted. By default, WebLogic Server does not allow null ciphers. See An Important Note Regarding Null Cipher Use in SSL.
InboundCertificateValidation	Specifies the client certificate validation rules for inbound SSL. This attribute applies only to a network channel that is configured to use two-way SSL. Either of the following values may be set: • `BuiltinSSLValidationOnly`—Uses the built-in trusted Certificate Authority-based validation. This is the default. • `BuiltinSSLValidationAndCertPathValidators`—Uses the built-in trusted CA-based validation and also the configured CertPathValidator providers to perform extra validation. For more information about these rules, see How SSL Certificate Validation Works in WebLogic Server. If a value for this attribute is not set, the value of the `SSLMBean.InboundCertificateValidation` attribute is used by default.
OutboundCertificateValidation	Specifies the server certificate validation rules for outbound SSL. Either of the following values may be set: • `BuiltinSSLValidationOnly`—Uses the built-in trusted Certificate Authority-based validation. This is the default. • `BuiltinSSLValidationAndCertPathValidators`—Uses the built-in trusted CA-based validation and also the configured CertPathValidator providers to perform extra validation. For more information about these rules, see How SSL Certificate Validation Works in WebLogic Server. If a value for this attribute is not set, the value of the `SSLMBean.OutboundCertificateValidation` attribute is used by default.

Steps to Configure a Channel-Specific Identity Keystore

You can configure a network channel to use a custom identity keystore different from the one used by the Managed Server configured in the WebLogic Remote Console.

To configure a channel-specific identity keystore, complete the following steps:

1. Configure a custom identity keystore, add the private key and the public identity certificate to be used by the network channel, and assign a private key alias.

- For information about configuring a JKS or PKCS12 keystore, see Configuring Keystores.
- For information about configuring an Oracle OPSS Key Store Service (KSS) keystore, see Configuring the OPSS Keystore Service for Custom Identity and Trust: Main Steps.

2. Create a custom network channel and assign the following attributes, ensuring that the combination of them is unique in the domain:
   - Channel name
   - Listen address
   - Listen port
   - Secure communication protocol (that is, either HTTPS or t3s)

   See Configure Custom Network Channels in *Oracle WebLogic Remote Console Online Help*.
3. Configure the channel to use the custom identity keystore created in Step 1 by setting the following attributes on the `NetworkAccessPointMBean`:
   - `CustomIdentityKeyStoreFileName` — If you are using a JKS or PKCS12 keystore, specify the path to the keystore. If you are using a KSS keystore, specify this value as the KSS URI.
   - `CustomIdentityKeyStoreType` — Specify the key store type. For example, JKS, PKCS12, or KSS.
   - Either the `CustomIdentityKeyStorePassPhraseEncrypted` attribute, or the `CustomIdentityKeyStorePassPhrase` attribute using the custom identity keystore passphrase.
   - `ChannelIdentityCustomized` — Set to `true`.
   - `CustomPrivateKeyAlias` — Specifies the string alias used to store and retrieve the channel's private key in the custom identity keystore. This private key is associated with the channel's identity certificate. Setting this attribute ensures that the channel alias corresponds to the channel's custom identity keystore and not to an alias in the server's identity keystore.
   - `CustomPrivateKeyPassPhrase` — Specify the value of the passphrase of the private key referenced by the `CustomPrivateKeyAlias` attribute.

> **Note:**
>
> If *any* of the `CustomIdentityKeyStoreFileName`, `CustomIdentityKeyStoreType`, `CustomIdentityKeyStorePassPhraseEncrypted`, or `CustomIdentityKeyStorePassPhrase` attributes are set, then all the following conditions must be met to ensure that the channel alias relates to the channel's custom identity keystore and does not default to an alias in the server keystore:
>
> a. *All* the preceding attributes must be set (that is, `CustomIdentityKeyStoreFileName`, `CustomIdentityKeyStoreType`, `CustomIdentityKeyStorePassPhraseEncrypted`, and `CustomIdentityKeyStorePassPhrase` must all be set).
>
> b. The `NetworkAccessPointMBean.ChannelIdentityCustomized` attribute must be set to `true`.
>
> c. The `NetworkAccessPointMBean.CustomPrivateKeyAlias` attribute must be set.
>
> Note that if *none* of the `CustomIdentityKeyStoreFileName`, `CustomIdentityKeyStoreType`, `CustomIdentityKeyStorePassPhraseEncrypted`, and `CustomIdentityKeyStorePassPhrase` attributes are set, the network channel's private key alias may be set to refer to the server keystore.

4. Configure any additional attributes for the network channel, as appropriate. See Configuring a Channel in *Administering Server Environments for Oracle WebLogic Server* and Configure Custom Network Channels in *Oracle WebLogic Remote Console Online Help*.

   For information about specifying a host name verifier class, see Using Host Name Verification.

   For information about inbound and outbound certificate validation, see SSL Certificate Validation.

Using WLST to Configure a Channel-Specific Identity Keystore

You can WLST to configure a network channel to use a custom identity keystore different from the one used by the Managed Server.

This section provides an example of using WLST to configure a channel-specific JKS identity keystore. Example 40-1 shows the following:

1. Connecting to a Managed Server instance.
2. Navigating to the MBean that corresponds to the specific network channel for which a custom identity keystore is to be configured, `https-override`.
3. Setting the name and location of the custom identity keystore file, `channelIdentity.jks`.
4. Setting the passphrase for the custom identity keystore.
5. Setting the custom identity keystore type to `JKS`.
6. Establishing that the channel's custom identity should be used.
7. Setting the custom private key alias to `myID`.
8. Setting the custom private key passphrase.

9. Saving and activating the new channel configuration, then disconnecting from the Managed Server instance.

> **Note:**
>
> You can also use a PKCS12 keystore. If you do so, be sure to use the `setCustomIdentityKeystoreType` property to set the keystore type to PKCS12.

Example 40-1 Configuring a Custom Identity Keystore

```
connect('','','t3://host:port')
Please enter your username :
Please enter your password :
...
edit()
startEdit()
cd ('Servers/myserver/NetworkAccessPoints/https-override')

cmo.setCustomIdentityKeyStoreFileName('/path/keystores/channelIdentity.jks')
cmo.setCustomIdentityKeyStorePassPhrase('passphrase')
cmo.setCustomIdentityKeyStoreType('JKS')
cmo.setChannelIdentityCustomized(true)
cmo.setCustomPrivateKeyAlias('myID')
cmo.setCustomPrivateKeyPassPhrase('keypassphrase')

save()
activate()
disconnect()
```

41

Configuring RMI over IIOP with SSL

Use SSL to protect Internet Interop-Orb-Protocol (IIOP) connections to Remote Method Invocation (RMI) remote objects in Oracle WebLogic Server. SSL secures connections through authentication and encrypts the data exchanged between objects.
To use SSL to protect RMI over IIOP connections:

1. Configure WebLogic Server to use SSL.
2. Configure the client Object Request Broker (ORB) to use SSL. Refer to the product documentation for your client ORB for information about configuring SSL.
3. Use the host2ior utility to print the WebLogic Server IOR to the console. The host2ior utility prints two versions of the interoperable object reference (IOR), one for SSL connections and one for non-SSL connections. The header of the IOR specifies whether or not the IOR can be used for SSL connections.
4. Use the SSL IOR when obtaining the initial reference to the CosNaming service that accesses the WebLogic Server JNDI tree.

For more information about using RMI over IIOP, see *Developing RMI Applications for Oracle WebLogic Server*.

42

Using a Certificate Callback Handler to Validate End User Certificates

Oracle WebLogic Server provides a means to examine details about information (such as the end user's certificate, Subject, and IP address) passed by an end user issuing a request to determine whether authentication should succeed or fail. This capability is provided by the `weblogic.security.SSL.CertificateCallback` interface, which you can implement to create a certificate callback handler.
When configured with WebLogic Server, this callback handler is invoked automatically whenever a client request is received over a secure RMI connection; for example, one that uses the T3s or IIOPS protocols. To configure a certificate callback handler so that it is in effect for all secure inbound RMI connections, you define it as a WebLogic Server system property that is passed in the server startup command.

This chapter includes the following topics:

How End User Certificate Callback Handlers Work

When a client makes a secure RMI connection to a WebLogic Server instance that is configured with a certificate callback handler, WebLogic Server invokes the callback handler. The callback evaluates details about the end user that are contained in the connection request, then returns a boolean value indicating whether authentication is successful.

The CertificateCallback interface calls the `validate` method on an `CertificateCallbackInfo` instance, which contains methods to obtain the following information from the end user that is contained in the RMI connection request:

- Client host name, IP address, and port
- Client domain name
- Destination host name, IP address, and port
- Authenticated Subject
- Client certificate

The callback implementation includes the logic that evaluates the client data that is obtained and returns `true` or `false` as follows:

- If the callback returns `true`, authentication succeeds and the client connection to WebLogic Server is made.
- If the callback returns `false`, a RemoteException is thrown containing the "Authentication denied" message.

> **Note:**
>
> If you use a certificate callback implementation in WebLogic Server, a callback is generated whenever a request is received over a secure port. As a result, using certificate callbacks may impose a performance overhead that should be taken into consideration.

Creating a Certificate Callback Implementation

The `weblogic.security.SSL.CertificateCallback` interface contains a single invocation on the `validate` method on a `weblogic.security.SSL.CertificateCallbackInfo` instance. The CertificateCallbackInfo instance contains methods to obtain details about the end user that are passed over the secure RMI connection.

You implement logic that evaluates the data that is returned and returns a `true` or `false`. The logic does not need to evaluate all data that is returned. Typically, only the certificate is evaluated; for example, obtaining the common name (cn) or distinguished name (dn).

See the following Javadoc in *Java API Reference for Oracle WebLogic Server*:

- `weblogic.security.SSL.CertificateCallback` interface
- `weblogic.security.SSL.CertificateCallbackInfo` class

Configuring the Certificate Callback with WebLogic Server

To configure the callback with WebLogic Server, specify the callback implementation as a system property in the WebLogic Server start command. The property should point to the callback implementation class that is on the server's classpath.

For example, if the callback implementation class is `MyCertificateCallback.java` in the package `com.mycompany.security`, and `MyCertificateCallback.class` is in the server's classpath, the following command sets the callback implementation property in WebLogic Server:

```
java weblogic.Server -
Dweblogic.security.SSL.CertificateCallback=com.mycompany.security.MyCertificateCallback
```

Note that if WebLogic Server is configured for one-way SSL, a client certificate is never sent to the server. Oracle recommends using certificate callbacks handlers only when WebLogic Server is configured for two-way SSL. See Set Up TLS in *Oracle WebLogic Remote Console Online Help*.

Part VII
Advanced Security Topics

Learn about the advanced security configuration options available in OracleWebLogic Server, such as cross-domain security and JASPIC security.

This part contains the following chapters:

43

Configuring Cross-Domain Security

Cross-domain security establishes trust between two WebLogic domain pairs by using a credential mapper to configure communication between these WebLogic domains. Learn how to set security configuration options that enables cross-domain security in Oracle WebLogic Server.
These sections apply to WebLogic Server deployments using the security features in this release of WebLogic Server.

- Enabling Trust Between WebLogic Server Domains
- Using the Java Authorization Contract for Containers
- Viewing MBean Attributes
- Configuring a Domain to Use JAAS Authorization

> **Note:**
>
> In this release of WebLogic Server, subsystems such as JMS, JTA, MDB, and WAN replication implement cross-domain security. These subsystems can authenticate and send the required credentials across domains. However, the EJB container does not implement the solution for cross-domain security.

Enabling Trust Between WebLogic Server Domains

WebLogic Server supports cross-domain security that establishes trust between two domains such that principals in a subject from one WebLogic domain can make calls in another domain. WebLogic Server establishes a security role for cross-domain users, and uses the WebLogic Credential Mapping security provider in each domain to store the credentials to be used by the cross-domain users.

Previous releases of WebLogic Server supported domain trust, which is now referred to as global trust. Global trust is established between two or more domains by using the same domain credential in each domain. If you enable global trust between two or more domains, the trust relationship is transitive and symmetric. In other words, if Domain A trusts Domain B, and Domain B trusts Domain C, then:

- Domain A will also trust Domain C.
- Domain B and Domain C will both trust Domain A.

The principal distinction between the two approaches is that cross-domain security enables trust between two domains using specific credentials. By comparison, in global trust, the same credentials are used to communicate among all domains.

In most cases, the cross-domain security approach is preferable to the global trust approach, because its use of a specific user group and role for cross-domain actions allows for finer-grained security.

> **Note:**
>
> If you enable cross-domain security to communicate between two domains, you should not enable global trust for those domains.
>
> Cross-domain security provides more secure communication between two domains.

The following sections explain how to configure each domain trust type:

- Enabling Cross-Domain Security Between WebLogic Server Domains
- Enabling Global Trust

Enabling Cross-Domain Security Between WebLogic Server Domains

> **Note:**
>
> In this release of WebLogic Server, subsystems such as JMS, JTA, MDB, and WAN replication implement cross-domain security. These subsystems can authenticate and send the required credentials across domains. However, the EJB container does not implement the solution for cross-domain security.

Configuration and use of cross-domain security is described in the following sections:

- Configuring Cross-Domain Security
- Excluding Domains From Cross-Domain Security
- Configuring Cross-Domain Users
- Configure a Credential Mapping for Cross-Domain Security

Configuring Cross-Domain Security

You configure cross-domain security between two domains — a domain pair — such that principals in a subject from one WebLogic domain can make calls in another domain. You can enable cross-domain security for multiple domain pairs.

For example, assume you have four domains, `Domain1` through `Domain4`. You can enable cross-domain security on all four domains, and then add users and credential maps (as described in subsequent sections) for the following domain pairs:

- `Domain1 - Domain2`
- `Domain1 - Domain3`
- `Domain1 - Domain4`
- `Domain2 - Domain3`
- `Domain2 - Domain4`
- `Domain3 - Domain4`

To configure cross-domain security in a WebLogic domain, set the `SecurityConfigurationMBean.CrossDomainSecurityEnabled` attribute to `true`. To do this in

WebLogic Remote Console, see Enable Cross Domain Security in *Oracle WebLogic Remote Console Online Help*.

Excluding Domains From Cross-Domain Security

If you enable cross-domain security for some, but not all, of the domains you administer, you need to add the names of the domains for which cross-domain security is not enabled to the list of excluded domains in the `SecurityConfigurationMBean.ExcludedDomainNames` attributes.

You must do this in each of the WebLogic domains in which you did enable cross-domain security.

For example, if you have four domains, `Domain1` through `Domain4` and for some reason you do not enable cross-domain security on `Domain4`, you need to specify `Domain4` for the `SecurityConfigurationMBean.ExcludedDomainNames` attribute in `Domain1`, `Domain2`, and `Domain3`.

To do this using WebLogic Remote Console:

1. In the **Edit Tree**, go to **Environment**, then **Domain**.
2. On the **Security** tab, in the **Excluded Domain Names** field, enter the names of any domains that do not have cross-domain security enabled. Enter the names of these domains separated either by semicolons or line breaks.
3. Repeat steps one through three, as appropriate, for each domain.
4. Click **Save**.

Configuring Cross-Domain Users

Cross-domain security in WebLogic Server uses a global security role named `CrossDomainConnector` with resource type `remote` and a group named `CrossDomainConnectors`. Invocation requests from remote domains are expected to be from users who are mapped to the `CrossDomainConnector` role.

By default, the `CrossDomainConnectors` group has no users as members.

For each domain in which you enable cross-domain security, you need to create a user and add that user to the `CrossDomainConnectors` group. Typically, such a user is a virtual system user and preferably should have no privileges other than those granted by the `CrossDomainConnector` security role.

For example, assume that you enabled cross-domain security on `Domain1`, `Domain2`, `Domain3`, and `Domain4`. In each case, create the user account with a password and assign it to the `CrossDomainConnectors` group.

- In `Domain1`, create a user `User1`.
- In `Domain2`, create `User2`.
- In `Domain3`, create `User3`.
- In `Domain4`, create `User4`.

To add a user in WebLogic Remote Console, see Create a User and Create a Group in *Oracle WebLogic Remote Console Online Help*.

Make sure that you add the users to the `CrossDomainConnectors` group.

Configure a Credential Mapping for Cross-Domain Security

> **Note:**
>
> The Credential Mapper identifies domains by their names. Therefore, it is important that the domains involved have unique names.

In the domain pairs for which you enabled cross-domain security, you need to specify a credential to be used by each user on the remote domain to be trusted. Do this by configuring credential mappings for each domain pair in the connection. Each credential mapping needs to specify:

- The resource protocol, which is named `cross-domain-protocol`
- The name of the remote domain that needs to interact with the local domain
- The name of the user in the remote domain that will be authorized to interact with the local domain
- The password of the user in the remote domain that will be authorized to interact with the local domain

For example, to extend the user example from Configuring Cross-Domain Users, you would configure the following domain pairs:

> **Note:**
>
> If you have a several domains to configure, you may find it easier to configure one pair of domains, then configure the next pair, and so forth.

- Populate the credential map in `Domain1` with the remote-domain: `Domain2`, the remote-user: `User2`, and the remote_user_pass: *`password-for-User2`*.

  Populate the credential map in `Domain2` with the remote-domain: `Domain1`, the remote-user: `User1`, and the remote_user_pass: *`password-for-User1`*.
- Populate the credential map in `Domain1` with the remote-domain: `Domain3`, the remote-user: `User3`, and the remote_user_pass: *`password-for-User3`*.

  Populate the credential map in `Domain3` with the remote-domain: `Domain1`, the remote-user: `User1`, and the remote_user_pass: *`password-for-User1`*.
- Populate the credential map in `Domain1` with the remote-domain: `Domain4`, the remote-user: `User4`, and the remote_user_pass: *`password-for-User4`*.

  Populate the credential map in `Domain4` with the remote-domain: `Domain1`, the remote-user: `User1`, and the remote_user_pass: *`password-for-User1`*.
- Populate the credential map in `Domain2` with the remote-domain: `Domain3`, the remote-user: `User3`, and the remote_user_pass: *`password-for-User3`*.

  Populate the credential map in `Domain3` with the remote-domain: `Domain2`, the remote-user: `User2`, and the remote_user_pass: *`password-for-User2`*.

- Populate the credential map in `Domain2` with the remote-domain: `Domain4`, the remote-user: `User4`, and the remote_user_pass: *`password-for-User4`*.

  Populate the credential map in `Domain4` with the remote-domain: `Domain2`, the remote-user: `User2`, and the remote_user_pass: *`password-for-User2`*.
- Populate the credential map in `Domain3` with the remote-domain: `Domain4`, the remote-user: `User4`, and the remote_user_pass: *`password-for-User4`*.

  Populate the credential map in `Domain4` with the remote-domain: `Domain3`, the remote-user: `User3`, and the remote_user_pass: *`password-for-User3`*.

To configure a cross-domain security credential mapping in WebLogic Remote Console, see the credential mapping step in Enable Cross Domain Security in *Oracle WebLogic Remote Console Online Help*.

Enabling Global Trust

> **Note:**
>
> Enabling global trust between WebLogic domains has the potential to open the servers up to man-in-the-middle attacks. Great care should be taken when enabling trust in a production environment. Oracle recommends having strong network security such as a dedicated communication channel or protection by a strong firewall.
>
> In most cases, the credential mapper approach, described in Enabling Cross-Domain Security Between WebLogic Server Domains, is preferable to the global trust approach, because it is provides closer control over access.

WebLogic Server enables you to establish global trust between two or more domains. You do this by specifying the same domain credential for each of the domains. By default, the domain credential is randomly generated and therefore, no two domains will have the same domain credential.

If you want two WebLogic domains to interoperate, you need to replace the generated credential with a credential you select, and set the same credential in each of the domains. For configuration information, see Enable Global Trust Between Domains in *Oracle WebLogic Remote Console Online Help*.

If you enable global trust between two domains, the trust relationship is transitive and symmetric. In other words, if `Domain A` trusts `Domain B` and `Domain B` trusts `Domain C`, then `Domain A` will also trust `Domain C`, and `Domain B` and `Domain C` will both trust `Domain A`.

Global trust between domains is established so that principals in a Subject from one WebLogic domain are accepted as principals in another domain. When this feature is enabled, identity is passed between WebLogic domains over an RMI connection without requiring authentication in the second domain. (For example, log in to `Domain 1` as Joe. Make an RMI call to `Domain 2` and Joe is still authenticated). WebLogic Server signs principals with the domain credential as principals are created. When a Subject is received from a remote source, its principals are validated. (The signature is recreated and, if it matches, the remote domain has the same domain credential). If validation fails, an error is generated. If validation succeeds, the Principals are trusted as if they were created locally.

> **Note:**
>
> Any credentials in clear text are encrypted the next time the `config.xml` file is persisted to disk.

If you are enabling global trust between domains in a Managed Server environment, you must stop the Administration Server and all the Managed Servers in both domains and then restart them. If this step is not performed, servers that were not rebooted will not trust the servers that were rebooted.

Keep the following points in mind when enabling global trust between WebLogic domains:

- Because a domain will trust remote principals without requiring authentication, it is possible to have authenticated users in a domain that are not defined in the domain's authentication database. This situation can cause authorization problems.
- Any authenticated user in a domain can access any other domain that has trust enabled with the original domain without re-authenticating. There is no auditing of this login and group membership is not validated. Therefore, if Joe is a member of the Administrators group in the original domain where he authenticated, he is automatically a member of the Administrators group for all trusted domains to which he makes RMI calls.
- If `Domain 1` trusts both `Domain 1` and `Domain 3`, `Domain 1` and `Domain 3` now implicitly trust each other. Therefore, members of the Administrators Group in `Domain 1` are members of the Administrators group in `Domain 3`. This may not be a desired trust relationship.
- If you extended the `WLSUser` and `WLSGroup` principal classes, the custom principal classes must be installed in the server's classpath in all domains that share trust.

To avoid these issues, Oracle recommends that rather than enabling global trust between two domains, you should instead use the approach described in Enabling Cross-Domain Security Between WebLogic Server Domains.

Using the Java Authorization Contract for Containers

As of version 12.2.1, WebLogic Server supports the Java Authorization Contract for Containers (JACC) Standard, Version 1.5. JACC can replace the EJB and servlet container deployment and authorization provided by WebLogic Server. Configure WebLogic Server to use JACC by using the command-line utility.

When you configure a WebLogic domain to use JACC, EJB and servlet authorization decisions are made by the classes in the JACC framework. All other authorization decisions within WebLogic Server are still determined by the WebLogic Security Framework. For information about the WebLogic JACC provider, see Using the Java Authorization Contract for Containers in *Developing Applications with the WebLogic Security Service*.

You configure WebLogic Server to use JACC by specifying the following properties in the command that starts WebLogic Server:

```
-Djavax.security.jacc.PolicyConfigurationFactory.provider
-Djavax.security.jacc.policy.provider
-Dweblogic.security.jacc.RoleMapperFactory.provider
```

For more information about these specifying these properties, see Enabling the WebLogic JACC Provider in *Developing Applications with the WebLogic Security Service*.

Note that an Administration Server and all Managed Servers in a domain need to have the same JACC configuration. If you change the JACC setting on the Administration Server, you should shut down the Managed Server and reboot them with the same settings as the Administration Server to avoid creating a security vulnerability. Otherwise, it may appear that EJBs and servlets in your domain are protected by WebLogic Security Framework roles and policies, when in fact the Managed Servers are still operating under JACC.

Viewing MBean Attributes

Use the `SecurityConfigurationMBean.AnonymousAdminLookupEnabled` attribute to control whether anonymous, read-only access should be allowed to WebLogic Server MBeans from the MBean API.

The **Anonymous Admin Lookup Enabled** option in the specifies whether anonymous, read-only access to WebLogic Server MBeans should be allowed from the MBean API. With this anonymous access, you can see the value of any MBean attribute that is not explicitly marked as protected by the WebLogic Server MBean authorization process. This option is enabled by default to assure backward compatibility. For greater security, you should disable this anonymous access.

Configuring a Domain to Use JAAS Authorization

The security configuration in a WebLogic domain can be modified to use JAAS authorization, which interprets Subjects differently from the way in which the WebLogic Security Service does.

When a principal requests access to a resource that is protected by the Java policy provider in Oracle Platform Security Services (OPSS), the principal is compared to another principal that is built from a name contained in a policy store. (This comparison occurs when the `Principal.equals()` method is invoked.) If the appropriate attributes of the two principal objects match, access is granted.

Principal comparison is not used by the WebLogic Security Service to determine access decisions to protected resources. However, when principal comparison is performed in a default WebLogic domain, the comparison of principal names is case sensitive, and only the names of the principals are compared. To use JAAS authorization, the security configuration of a WebLogic domain can be modified to accommodate the following principal comparison behavior:

- The comparison of principal names is case insensitive
- The GUID and DN data in WebLogic principal objects are included in the comparison

To modify the security configuration of a WebLogic domain so that principal objects can be used with JAAS authorization, the following MBean attributes settings are available:

```
SecurityConfigurationMBean.PrincipalEqualsCaseInsensitive="true"
SecurityConfigurationMBean.PrincipalEqualsCompareDnAndGuid="true"
```

To set these attributes in WebLogic Remote Console:

1. In the **Edit Tree**, go to **Environment**, then **Domain**.
2. On the **Security** tab, click **Show Advanced Fields**.
3. Turn on the **Use Case-insensitive Principal Name Matching** option.
4. Turn on the **Use LDAP DN & GUID in Principal Matching** option.
5. Click **Save**.

> **Note:**
>
> If a domain is configured to use the GUID and DN data in principals, there may be an impact when interoperating with other WebLogic domains, particularly older domains, resulting from changes made to the way identity is passed.

For information about principal comparison in the Oracle Platform Security Service, see Principal Name Comparison Logic in *Securing Applications with Oracle Platform Security Services*.

For information about passing identity to a WebLogic domain, see *Developing Standalone Clients for Oracle WebLogic Server*.

44

Configuring JASPIC Security

The Java Authentication Service Provider Interface for Containers (JASPIC) specification defines a service provider interface (SPI). The JASPIC SPI is used by authentication providers that implement message authentication mechanisms that can be integrated in server Web application message processing. Learn how to configure JASPIC security in Oracle WebLogic Server.
Read the JASPIC specification at http://www.jcp.org/en/jsr/detail?id=196.

This chapter includes the following sections:

- JASPIC Mechanisms Override WebLogic Server Defaults
- Prerequisites for Configuring JASPIC
- Location of Configuration Data
- Configuring JASPIC for a Domain
- Configuring JASPIC Using WLST

This section assumes that you are familiar with a basic overview of JASPIC, as described in JASPIC Security in *Understanding Security for Oracle WebLogic Server*.

JASPIC Mechanisms Override WebLogic Server Defaults

If you configure an Authentication Configuration Provider for a Web application, it is used instead of the WebLogic Server authentication mechanism for that Web Application. The JASPIC authentication provider assumes responsibility for authenticating the user credentials and returning a Subject.

You should therefore exercise care when you specify an Authentication Configuration Provider to make sure that it satisfies your security authentication needs.

Prerequisites for Configuring JASPIC

There are certain prerequisites for configuring JASPIC in your environment including, how to make your own or third party server authentication module (SAM) or Authentication Configuration Providers available to WebLogic Server.

The JASPIC programming model is described in the Java Authentication Service Provider Interface for Containers (JASPIC) specification (http://www.jcp.org/en/jsr/detail?id=196).

A sample SAM implementation is described in Adding Authentication Mechanisms to the Servlet Container in the *GlassFish Server Open Source Edition Application Development Guide*. Although written from the GlassFish Server perspective, the tips for writing a SAM, and the sample SAM itself, are instructive.

This section includes the following topics:

- Server Authentication Module Must Be in Classpath
- Custom Authentication Configuration Providers Must Be in Classpath

Server Authentication Module Must Be in Classpath

If you plan to configure a WebLogic Server Authentication Configuration Provider, you must add the jar for your SAM to the system classpath via the startup scripts or the command line used to start the WebLogic Server instance. If you do not do this, WebLogic Server is not able to find the appropriate classes.

Custom Authentication Configuration Providers Must Be in Classpath

If you plan to configure a custom Authentication Configuration Provider, you must add the jar for your custom Authentication Configuration Provider to the system classpath via the startup scripts or the command line used to start the WebLogic Server instance. If you do not do this, WebLogic Server is not able to find the appropriate classes.

Location of Configuration Data

You can use the WebLogic Scripting Tool (WLST) to configure JASPIC and the Authentication Configuration Providers. After you configure JASPIC and the Authentication Configuration Providers, the domain-wide Authentication Configuration Provider configuration data is kept in the domain `config.xml` file in the `<jaspic>` element.

For example:

```
<jaspic>
      <auth-config-provider xsi:type="wls-auth-config-providerType">
        <name>WLSAuthConfigProvider-0</name>
      </auth-config-provider>
    </jaspic>
```

When you configure an Authentication Configuration Provider for a deployed Web application, WLST updates the deployment plan (plan.xml) for the Web application with the application-specific Authentication Configuration Provider configuration. For example:

```
<variable>
      <name>JASPICProvider_AuthConfigProviderName_13210476440805</name>
      <value>WLSAuthConfigProvider-0</value>
</variable>
:
<variable-assignment>
   <name>JASPICProvider_AuthConfigProviderName_13210476440805</name>
   <xpath>/weblogic-web-app/jaspic-provider/auth-config-provider-name</xpath>
</variable-assignment>
```

If you do not use a deployment plan for your application, you can instead add the `jaspic-provider` deployment descriptor element to `weblogic.xml`.

`jaspic-provider` specifies the `authConfigProvider` to be registered for use during authentication. For example, `<wls:jaspic-provider>my-acp</wls:jlaspic-provider>`.

Configuring JASPIC for a Domain

You can configure JASPIC for a domain using WebLogic Remote Console and WLST.

By default, JASPIC is enabled for a domain.

If you disable JASPIC for a domain, then JASPIC is disabled for all Web applications in that domain, regardless of their configuration.

To configure JASPIC for a domain:

1. In WebLogic Remote Console, open the **Edit Tree** and go to **Environment**, then **Domain**.
2. On the **Security** tab, click **Show Advanced Fields**.
3. Turn on the **JASPIC Enabled** option.
4. Click **Save** and commit your changes.
5. Using WLST, configure Authentication Configuration providers. See Configuring JASPIC Using WLST.

After you configure JASPIC properties for the domain, you can specify which Authentication Configuration provider applies to a specific Web application. See Configure Web Applications for JASPIC in *Oracle WebLogic Remote Console Online Help*.

Configuring JASPIC Using WLST

You can use WLST to configure JASPIC for a domain, and perform tasks such as creating a WLS Authentication Configuration Provider or a custom Authentication Configuration Provider, listing all WLS and custom Authentication Configuration Providers, enabling and disabling JASPIC for a domain.

For information about using WLST, see *Understanding the WebLogic Scripting Tool*.

This section requires you to configure the following MBeans via WLST:

- JASPICMBean
- CustomAuthConfigProviderMBean
- WLSAuthConfigProviderMBean

See *MBean Reference for Oracle WebLogic Server* for additional MBean information.

Creating a WLS Authentication Configuration Provider

Example 44-1 creates a WLS Authentication Configuration Provider, sets the class name of the SAM, and sets a configuration property.

After you run this example, restart WebLogic Server.

Example 44-1 Create a WLS Authentication Configuration Provider

```
connect('','','t3://host:port')
Please enter your username :
Please enter your password :
...
edit()
startEdit()
cd('SecurityConfiguration')
cd('mydomain')
jaspic = cmo.getJASPIC()
wacp = jaspic.createWLSAuthConfigProvider('wacp')
am = wacp.getAuthModule()
am.setClassName('com.my.auth.module.Classname')
props = Properties()
props.setProperty('property', 'value')
am.setProperties(props)
```

```
save()
activate()
```

Creating a Custom Authentication Configuration Provider

Example 44-2 creates a custom Authentication Configuration Provider, sets the class name of this Authentication Configuration Provider, and sets a configuration property.

After you run this example, restart WebLogic Server.

Example 44-2 Create a Custom Authentication Configuration Provider

```
connect('','','t3://host:port')
Please enter your username :
Please enter your password :
...
edit()
startEdit()
cd('SecurityConfiguration')
cd('mydomain')
jaspic = cmo.getJASPIC()
acp = jaspic.createCustomAuthConfigProvider('cacp')
acp.setClassName('com.my.acp.Classname')
props = Properties()
props.setProperty('property', 'value')
acp.setProperties(props)
save()
activate()
```

Listing All WLS and Custom Authentication Configuration Providers

Example 44-3 shows how to list all Authentication Configuration Providers for a domain.

Example 44-3 List All Authentication Configuration Providers

```
connect('','','t3://host:port')
Please enter your username :
Please enter your password :
...
edit()
startEdit()
cd('SecurityConfiguration')
cd('mydomain')
jaspic = cmo.getJASPIC()
jaspic.getAuthConfigProviders()
```

Enabling JASPIC for a Domain

Example 44-4 shows how to enable JASPIC for a domain.

After you run this example, restart WebLogic Server.

Example 44-4 Enable JASPIC for a Domain

```
connect('','','t3://host:port')
Please enter your username :
Please enter your password :
...
edit()
startEdit()
cd('SecurityConfiguration')
```

```
cd('mydomain')
jaspic = cmo.getJASPIC()
jaspic.setEnabled(false)
save()
activate()
```

Disabling JASPIC for a Domain

Example 44-5 shows how to disable JASPIC for a domain.

After you run this example, restart WebLogic Server.

Example 44-5 Disable JASPIC for a Domain

```
connect('','','t3://host:port')
Please enter your username :
Please enter your password :
...
edit()
startEdit()
cd('SecurityConfiguration')
cd('mydomain')
jaspic = cmo.getJASPIC()
jaspic.setEnabled(false)
save()
activate()
```

45

Using the Java EE Security API in WebLogic Server

Using the Java EE Security API, you can define all of the security information directly within the application. Bundling the security configuration in the application instead of configuring it externally improves the management of the application's lifecycle, especially in a world of microservices that are distributed in containers.

Overview of the Java EE Security API (JSR 375)

The Java EE Security API (JSR 375) defines portable authentication mechanisms, and an access point for programmatic security using the `SecurityContext` interface. In WebLogic Server, these authentication mechanisms are supported in the web container, and the `SecurityContext` interfaces are supported in the Servlet and EJB containers.

WebLogic Server supports the plug-in interface for authentication, `HttpAuthenticationMechanism`, and includes built-in support for the BASIC, FORM, and Custom FORM authentication mechanisms defined in the specification. WebLogic Server also supports the `RememberMeIdentityStore` interface, and built-in implementations of the `IdentityStore` interface (LDAP identity store and Database identity store) as well as the custom identity store.

The `HttpAuthenticationMechanism` interface is designed to capitalize on the strengths of existing Servlet and JASPIC authentication mechanisms. The `IdentityStore` interface is intended primarily for use by `HttpAuthenticationMechanism` implementations, but could in theory be used by other types of authentication mechanisms (such as a JASPIC `ServerAuthModule`). `HttpAuthenticationMechanism` implementations are not required to use `IdentityStore` — they can authenticate users in any manner they choose — but the `IdentityStore` interface is a useful and convenient mechanism.

The `HttpAuthenticationMechanism` and `IdentityStore` interfaces are implemented as CDI beans, therefore they are visible to the container through CDI. For information on CDI support in WebLogic Server, see Using Contexts and Dependency Injection for the Java EE Platform in *Developing Applications for Oracle WebLogic Server*.

The `SecurityContext` interface defines methods that allow an application to access security information about a caller, authenticate a caller, and authorize a caller.

The programming model for the Java EE Security API 1.0 (JSR 375) is defined in the specification at `https://www.jcp.org/en/jsr/detail?id=375`.

For details about using JSR 375 in WebLogic Server, see Using the Java EE Security API in *Developing Applications with the WebLogic Security Service*.

Prerequisites for Using the Java EE Security API

Using the Java EE Security API mechanisms does not require any specific configuration, but you must ensure that other functionality, such as JASPIC and CDI, is enabled.

To use the Java EE Security API (JSR 375) features in WebLogic Server:

- JASPIC must be enabled at the domain level to enable JSR 375 functionality. By default, JASPIC is enabled for a domain in WebLogic Server. If you disable JASPIC at the domain level, JSR 375 functionality is also disabled.
- Web applications must include the `beans.xml` deployment descriptor file in the application's WAR or EAR file, as specified by the CDI specification (`https://jcp.org/en/jsr/detail?id=365)`.
- The `metadata-complete` attribute in the `web.xml` file for the web applications must NOT be set to `true`. The default in WebLogic Server is `false`.
- There are no special logging requirements. Audit events triggered by implementations of the Java EE Security API are logged by the WebLogic Auditing Provider, if configured.
- The Java EE Security API requires that group principal names are mapped to roles of the same name by default. If the `security-role-assignment` element in the `weblogic.xml` deployment descriptor does not declare a mapping between a security role and one or more principals in the WebLogic Server security realm, then the role name is used as the default principal.

46

Using Secured Production Mode

In a WebLogic Server domain, the domain mode determines the default values to apply to the security configuration of the domain. Secured production mode applies the strictest default values to the security configuration of your domain.

The domain modes, in order from least to most secure default values, are:

- Development mode
- Production mode
- Secured production mode

When you enable secured production mode, WebLogic Server automatically sets some security configurations to more secure values. However, there are certain security configurations that require additional configuration.

The domain mode only specifies the *default* values of a domain's security configuration. You can still modify individual configurations to override the default values. Overriding default values can help you fine tune your configuration to meet functional and security requirements, but overriding secured production mode default values should be done with caution, as it may result in a less secure environment.

If your domain does not meet the security criteria of a domain mode, WebLogic Server will flag any insecure values and report them as security warnings. For more information on security validation warnings, see Review Potential Security Issues in *Securing a Production Environment for Oracle WebLogic Server*. You may also experience behavioral issues when trying to manage your domain.

For information on the default values in each domain mode, see Understand How Domain Mode Affects the Default Security Configuration in *Securing a Production Environment for Oracle WebLogic Server*.

Although secured production mode can be a powerful tool for securing your domain, it is limited by the complexity and sheer variety of WebLogic Server environments. To ensure your domain is well protected, you should also review the recommendations outlined in Configuring Security for a WebLogic Domain and in *Securing a Production Environment for Oracle WebLogic Server* and then apply them as appropriate to your environment.

Changes to the Domain Mode

The domain mode is specified as part of the initial domain configuration process. Although it is possible to change the domain mode after a domain is created, interactions between an existing domain configuration and the configurations that are applied by the new domain mode can lead to unexpected outcomes.

The domain mode only controls default values, therefore any configurations that you have set explicitly will persist after a change in domain mode and supersede the values that would otherwise be used. Additionally, if the default value of a configuration in the new domain mode matches the existing, explicit value, it will remove the explicit configuration (and its value) from the domain configuration. Then, if you decide to revert to your previous domain mode, the explicit configuration will remain absent.

Before you change the domain mode to secured production mode, make sure that you carefully review your domain configuration file, `config.xml`, and compare its existing values to the MBean attributes that are determined by domain mode. See Secure Values for MBean Attributes in *MBean Reference for Oracle WebLogic Server*. To change the domain mode after domain creation, see Changing the Domain Mode.

When is Secured Production Mode Enabled?

As of WebLogic 14.1.2.0.0, when you set the domain mode to production mode, it enables secured production mode by default. In previous releases, when you enabled production mode, secured production mode was disabled by default and you had to enable it explicitly.

If you upgrade from WebLogic Server 14.1.1.0.0 and earlier, the behavior of your domain mode will not change. For example, when a domain in production mode is upgraded from 14.1.1.0.0 to 14.1.2.0.0 or later, it will remain in production mode with secured production mode *disabled*. However, if you upgrade your non-production mode domain to 14.1.2.0.0 or later, and *then* change the domain mode to production mode, it will *enable* secured production mode by default.

> **Note:**
>
> You can still use production mode with secured production mode disabled, but you must explicitly disable secured production mode. See Change the Domain Mode in *Oracle WebLogic Remote Console Online Help*.

The domain configuration file, `config.xml`, does not explicitly state that secured production mode is enabled because secured production mode enabled is now the default state of production mode.

Table 46-1 Domain Mode in the Domain Configuration File (config.xml)

WebLogic Server Release	Secured Production Mode is Enabled	Secured Production Mode is Disabled

14.1.2.0.0 and later	`<production-mode-enabled>true</production-mode-enabled>`	`<production-mode-enabled>true</production-mode-enabled>`
		`<secure-mode>`
		`  <secure-mode-enabled>false</secure-mode-enabled>`
		`</secure-mode>`

Table 46-1 (Cont.) Domain Mode in the Domain Configuration File (config.xml)

WebLogic Server Release	Secured Production Mode is Enabled	Secured Production Mode is Disabled

14.1.1.0.0 and earlier	`<production-mode-enabled>true</production-mode-enabled>`
	`<secure-mode>`
	`<secure-mode-enabled>true</secure-mode-enabled>`
	`</secure-mode>`
		`<production-mode-enabled>true</production-mode-enabled>`

Changing the Domain Mode

You can change the domain mode on an existing domain.

Changes to the domain mode require a full domain restart - a rolling restart is not sufficient. Oracle recommends that you use offline tools (such as WLST Offline) to modify the domain mode. If you modify the domain mode on running domains, using tools such as WebLogic Remote Console, you must still shut down all of the servers in the domain and then restart them, one server at a time.

When changing the domain mode of an existing domain, consider saving the existing `config.xml` file so you can compare it with the new `config.xml` file after the domain mode change and assess which settings were changed. See Changes to the Domain Mode for an explanation of the potential effects of making domain mode changes on an existing domain.

- If using WLST Offline:
  1. Shut down the domain.
  2. Invoke WLST Offline. See Invoking WLST in *Understanding the WebLogic Scripting Tool*.
  3. Run the WLST Offline script that changes your current domain mode to your target domain mode.

Current Domain Mode	Target Domain Mode	WLST Offline Script

Development	Production
	Note: Secured Production Mode is enabled by default.
		`readDomain('DOMAIN_NAME')`
		`cmo.setProductionModeEnabled(true)`
		`updateDomain()`

Current Domain Mode	Target Domain Mode	WLST Offline Script

Development	Production (with Secured Production Mode explicitly disabled)	`readDomain('DOMAIN_NAME')`
		`cmo.setProductionModeEnabled(true)`
		`cd('/SecurityConfiguration/%s' %(cmo.getName()))`
		`create('NO_NAME','SecureMode')`
		`cd('SecureMode/NO_NAME')`
		`set('SecureModeEnabled','false')`
		`updateDomain()`

Production (with Secured Production Mode disabled)	Secured Production	`readDomain('DOMAIN_NAME')`
		`cd('/SecurityConfiguration/%s/SecureMode/NO_NAME_0' %(cmo.getName()))`
		`set('SecureModeEnabled',true)`
		`updateDomain()`

Production	Development	`readDomain('DOMAIN_NAME')`
		`cmo.setProductionModeEnabled(false)`
		`cmo.getSecurityConfiguration().getSecureMode().setSecureModeEnabled(false)`
		`updateDomain()`

Secured Production	Production (with Secured Production Mode explicitly disabled)	`readDomain('DOMAIN_NAME')`
		`cd('/SecurityConfiguration/%s' %(cmo.getName()))`
		`create('NO_NAME','SecureMode')`
		`cd('SecureMode/NO_NAME')`
		`set('SecureModeEnabled','false')`
		`updateDomain()`

Secured Production	Development	`readDomain('DOMAIN_NAME')`
		`cmo.setProductionModeEnabled(false)`
		`updateDomain()`

4. Start your domain.

- If using WebLogic Remote Console, see Change the Domain Mode in *Oracle WebLogic Remote Console Online Help*.

Changes to the domain mode can affect the default URL of the Administration Server. When SSL/TLS and the administration port are enabled (by default, both are enabled in secured production mode), the default URL is `https://hostname:9002` or `t3s://hostname:9002`. Take note of the protocol and the port number. When SSL/TLS and the administration port are disabled (by default, both are disabled in development mode), the default URL is `http://hostname:7001` or `t3://hostname:7001`.

When you enable secured production mode, WebLogic Server expects certain security configurations to be configured. If they are not configured properly, it will flag settings it deems

insecure and possibly block traffic on ports or addresses that were previously available in less secure domain modes.

Overriding the Domain Mode (Single Server Domains Only)

It is possible to override the current domain mode of your domain for the duration of its server life cycle. After you restart the server, it will return to its normal domain mode.

You should only perform this task on single server domains in development or test environments.

> **Note:**
>
> When you override the domain mode at the command line, you will not see the effective domain mode in the `config.xml` file or the WebLogic Remote Console **Edit Tree** perspective. Instead, you can use WLST or the **Configuration View Tree** perspective in WebLogic Remote Console to see the effective state of the domain. See Verifying Attribute Values That Are Set on the Command Line in *Command Reference for Oracle WebLogic Server*.

If you want to override the current domain mode of your domain, then run the Administration Server start script and include a system property that determines the effective domain mode. See Table 46-2 for the system properties and their usage.

For example, to try out production mode, run:

```
startWebLogic.sh -Dweblogic.ProductionModeEnabled=true
```

You can also set environment variables to achieve the same outcome. For example:

```
export DOMAIN_PRODUCTION_MODE="true"
startWebLogic.sh
```

Table 46-2 Overriding the domain mode at the command line

Current Domain Mode	Target Domain Mode	System Property	Environment Variable
Development	Production **Note**: Secured Production Mode is enabled by default.	`weblogic.ProductionModeEnabled=true`	`DOMAIN_PRODUCTION_MODE="true"`
Development	Production (with Secured Production Mode disabled)	Include both system properties: • `weblogic.ProductionModeEnabled=true` • `weblogic.securemode.SecureModeEnabled=false`	Set both environment variables: • `DOMAIN_PRODUCTION_MODE="true"` • `SECURE_PRODUCTION_MODE="false"`

Table 46-2 (Cont.) Overriding the domain mode at the command line

Current Domain Mode	Target Domain Mode	System Property	Environment Variable
Production (with Secured Production Mode disabled)	Secured Production	`weblogic.securemode.SecureModeEnabled=true`	`SECURE_PRODUCTION_MODE="true"`
Production	Development	`weblogic.ProductionModeEnabled=false`	`DOMAIN_PRODUCTION_MODE="false"`
Secured Production	Production (with Secured Production Mode disabled)	`weblogic.securemode.SecureModeEnabled=false`	`SECURE_PRODUCTION_MODE="false"`
Secured Production	Development	`weblogic.ProductionModeEnabled=false`	`DOMAIN_PRODUCTION_MODE="false"`

Connecting to the Administration Server using WebLogic Remote Console

Depending on your existing security settings, you may need to perform additional configuration before you can manage a domain with secured production mode enabled.

1. Choose one of the options for starting an Administration Server as described in Starting and Stopping Servers in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

   You may want to create a boot identity file to store user credentials for starting and stopping an instance of WebLogic Server. See Creating a Boot Identity File for an Administration Server in *Administering Server Startup and Shutdown for Oracle WebLogic Server*. If you choose to create a boot identity file, then make sure to set appropriate permissions on the `boot.properties` file and its containing folder, *`DOMAIN_HOME`*`/servers/AdminServer/security/`. We recommend setting `chmod 600` on `boot.properties` and `chmod 740` for `DOMAIN_HOME/servers/AdminServer/security`.

2. Connect to the domain using WebLogic Remote Console as described in Connect to an Administration Server in *Oracle WebLogic Remote Console Online Help*.

   The default address of your Administration Server is now `https://`*`hostname`*`:9002` . Note the `s` in *https* and the port number.

   In secured production mode, the non-SSL/TLS listen port (typically 7001) is disabled by default and traffic is routed over the SSL/TLS ports. If you want to configure listen ports, see Specify Listen Ports in *Oracle WebLogic Remote Console Online Help*.

3. Configure custom keystores as described in Configure Keystores in *Oracle WebLogic Remote Console Online Help*.

   In production environments, you should configure custom keystores for identity and trust. See Obtaining and Storing Certificates for Production Environments. If you want to use the demo certificates provided by WebLogic Server instead, review the topics under Secured Production Mode in Development Environments.

4. Make sure that SSL/TLS is configured properly, as described in Set Up TLS in *Oracle WebLogic Remote Console Online Help*.
5. Enable host name verification, as described in Enable Host Name Verification in *Oracle WebLogic Remote Console Online Help*.
6. Optional: If you use Oracle Platform Security Services (OPSS) but you want to configure your domain with *custom* certificates, see Replacing Demonstration CA Signed Certificates in *Securing Applications with Oracle Platform Security Services* instead.
7. Save and commit your changes.
8. Restart the Administration Server.

Starting Managed Servers using WebLogic Remote Console

If you use WebLogic Remote Console to start Managed Servers, you must configure Node Manager to register the properties for your custom keystore.

1. Ensure Node Manager is configured properly to work with WebLogic Remote Console, as described in Start Managed Servers in *Oracle WebLogic Remote Console Online Help*.
2. Update the `nodemanager.properties` file with the following attributes and their values:
   - `CustomIdentityAlias`
   - `CustomIdentityKeyStoreFileName`
   - `CustomIdentityPrivateKeyPassPhrase`
   - `CustomIdentityKeyStorePassPhrase`
   - `KeyStores`

   See Node Manager Properties in *Administering Node Manager for Oracle WebLogic Server*.
3. Optional: If multiple server instances run on the same computer in the domain and the domain-wide administration port is enabled, then you must perform one of the following:
   - Host the server instances on a multi-homed machine and assign each server instance a unique listen address
   - Override the domain-wide port on all but one of the servers instances on the machine. On the **Environment**: **Servers**: *myServer* page for each Managed Server, enter a unique port value in the **Local Administration Port Override** field.
4. Save and commit your changes.
5. Start the Managed Server.

Connecting to the Administration Server using WLST

You must perform additional configuration before you can use the WebLogic Scripting Tool to connect to a domain with secured production mode enabled.

1. Start the Administration Server.

   You may want to create a boot identity file to store user credentials for starting and stopping an instance of WebLogic Server. See Creating a Boot Identity File for an Administration Server in *Administering Server Startup and Shutdown for Oracle WebLogic Server*. If you choose to create a boot identity file, then make sure to set appropriate permissions on the `boot.properties` file and its containing folder, *`DOMAIN_HOME`*`/servers/`

AdminServer/security/. We recommend setting chmod 600 on boot.properties and chmod 740 for DOMAIN_HOME/servers/AdminServer/security.

2. Update the WLST_PROPERTIES environment variable to configure keystores:

```
export WLST_PROPERTIES="-Dweblogic.security.TrustKeyStore=CustomTrust -
Dweblogic.security.CustomTrustKeyStoreFileName=trustKeystoreFile -
Dweblogic.security.CustomTrustKeyStorePassPhrase=trustKeyStorePassword"
```

3. Connect to the domain using the WLST as described in Invoking WLST in *Understanding the WebLogic Scripting Tool*.

> **Note:**
>
> The default address of an Administration Server in secured production mode is t3s://*hostname*:9002. Note the s in t3s and the port number.

Starting Managed Servers using a Start Script

You can start Managed Servers using a start script.

1. Create a boot identity file for each Managed Server. See Creating Boot Identity Files for Managed Servers in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

   Make sure to set appropriate permissions on the file and its containing folder. We recommend setting chmod 600 on the boot.properties file and chmod 740 for the folder, *DOMAIN_HOME*/servers/*managedServerName*/security/. If a boot.properties file is not available, then you must include a username and password when starting servers from the command line.

2. Specify the keystore values for the Managed Server by adding the following arguments to the JAVA_OPTIONS environment variable.

   - -Dweblogic.security.SSL.trustedCAKeyStore
   - -Dweblogic.security.SSL.trustedCAKeyStorePassPhrase

   For example:

```
export JAVA_OPTIONS="-
Dweblogic.security.SSL.trustedCAKeyStore=trustKeystoreFile -
Dweblogic.security.SSL.trustedCAKeyStorePassPhrase=trustKeyStorePassword"
```

3. Run the startManagedWebLogic script as described in Starting Managed Servers with a Startup Script in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

   For example:

```
startManagedWebLogic.sh managedServerName https://adminHostname:adminPort
```

Stopping Servers

- To shut down a server instance from WebLogic Remote Console, see Stop a Server in *Oracle WebLogic Remote Console Online Help*.
- To shut down a server instance with a script, see Shutting Down Servers with a Stop Script in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

> **Note:**
>
> – If you have not added arguments for your keystores to the `JAVA_OPTIONS` environment variable, then you must include them when stopping the server.
> – If you do not have a `boot.properties` file configured, then you will need to include the username and password at the command line.
> – If you want to use HTTPS instead of T3s as the protocol in the domain URL , you must enable tunnelling and default internal servlets. Set `ServerMBean.TunnelingEnabled` to `true` and `ServerMBean.DefaultInternalServletsDisabled` to `false`.

For example:

```
export JAVA_OPTIONS="-
Dweblogic.security.SSL.trustedCAKeyStore=trustKeystoreFile -
Dweblogic.security.SSL.trustedCAKeyStorePassPhrase=trustKeyStorePassword"
# To stop a Managed Server
stopManagedWebLogic.sh managedServerName t3s://adminHostname:adminPort
[wlsUsername wlsPassword]
# To stop the Administration Server
export ADMIN_URL="t3s://adminHostname:adminPort"
stopWebLogic.sh [wlsUsername wlsPassword]
```

Secured Production Mode in Development Environments

If you want to assess the features of secured production mode but do not want to expend the effort to set up custom keystores, it is possible to configure WebLogic Server to use secured production mode with the demonstration keystores included with WebLogic Server.

For information on the demonstration keystores, see Using Keystores and Certificates in a Development Environment. If you want to use the OPSS Keystore Service, see also Configuring Oracle OPSS Keystore Service.

> **Note:**
>
> The following procedures are suitable for testing and development purposes only. Do not use demo keystores in a true production environment.

- Using Secured Production Mode with Demonstration Keystores

- Using Secured Production Mode with Demonstration Keystores with KSS

Using Secured Production Mode with Demonstration Keystores

You may need to perform additional configuration to support the use of the insecure demo keystores in domains with secured production mode enabled.

1. Set the domain mode to secured production mode using one of the following methods:
   - Create a new domain and select secured production mode as the domain mode when prompted.

     Creating a new domain allows you to configure many of the settings that are necessary for secured production mode in a single procedure and reduces the likelihood of conflicting configurations.

     See Creating a WebLogic Domain in *Creating WebLogic Domains Using the Configuration Wizard*.
   - Modify an existing domain to use secured production mode.

     See Changing the Domain Mode.
2. Choose one of the options for starting an Administration Server as described in Starting and Stopping Servers in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

   You may want to create a boot identity file to store user credentials for starting and stopping an instance of WebLogic Server. See Creating a Boot Identity File for an Administration Server in *Administering Server Startup and Shutdown for Oracle WebLogic Server*. If you choose to create a boot identity file, then make sure to set appropriate permissions on the `boot.properties` file and its containing folder, *`DOMAIN_HOME`*`/servers/AdminServer/security/`. We recommend setting `chmod 600` on `boot.properties` and `chmod 740` for `DOMAIN_HOME/servers/AdminServer/security`.
3. Connect to the domain using WebLogic Remote Console as described in Connect to an Administration Server in *Oracle WebLogic Remote Console Online Help*.

   The default address of your Administration Server is now `https://`*`hostname`*`:9002`. Note the `s` in *https* and the port number.

   In secured production mode, the non-SSL/TLS listen port (typically 7001) is disabled by default and traffic is routed over the SSL/TLS ports. If you want to configure listen ports, see Specify Listen Ports in *Oracle WebLogic Remote Console Online Help*.
4. Make sure that WebLogic Server is not using the demo keystores provided by the Oracle Platform Security Services (OPSS) Keystore Service (KSS).
   a. In WebLogic Remote Console, in the **Edit Tree**, go to **Environment**, then **Domain**.
   b. On the **Security** tab, click **Show Advanced Fields**.
   c. Turn off the **Use KSS for Demo** option.
   d. Click **Save**.
5. Make sure that SSL/TLS is configured properly, as described in Set Up TLS in *Oracle WebLogic Remote Console Online Help*.
6. Save and commit your changes and then restart the Administration Server.
7. If your domain includes Managed Servers, and you plan to use WebLogic Remote Console to start them, perform these steps too:

a. Ensure Node Manager is configured properly to work with WebLogic Remote Console, as described in Start Managed Servers in *Oracle WebLogic Remote Console Online Help*.

b. Add `UseKSSForDemo=False` to the `nodemanager.properties` file.

   See Node Manager Properties in *Administering Node Manager for Oracle WebLogic Server*.

c. Optional: If multiple server instances run on the same computer in the domain and the domain-wide administration port is enabled, then you must perform one of the following:

   - Host the server instances on a multi-homed machine and assign each server instance a unique listen address
   - Override the domain-wide port on all but one of the servers instances on the machine. On the **Environment**: **Servers**: *myServer* page for each managed server, enter a unique port value in the **Local Administration Port Override** field.

d. Save and commit your changes.

e. Start the Managed Server.

Using WLST on Domains using Demo Keystores

When using WLST, you must perform additional configuration to support the use of the insecure demo keystores in domains with secured production mode enabled.

1. Start the Administration Server.

   You may want to create a boot identity file to store user credentials for starting and stopping an instance of WebLogic Server. See Creating a Boot Identity File for an Administration Server in *Administering Server Startup and Shutdown for Oracle WebLogic Server*. If you choose to create a boot identity file, then make sure to set appropriate permissions on the `boot.properties` file and its containing folder, *`DOMAIN_HOME`*`/servers/AdminServer/security/`. We recommend setting `chmod 600` on `boot.properties` and `chmod 740` for `DOMAIN_HOME/servers/AdminServer/security`.

2. Update the `WLST_PROPERTIES` environment variable to configure keystores:

   ```
   export WLST_PROPERTIES="-Dweblogic.RootDirectory=DOMAIN_HOME -
   Dweblogic.security.TrustKeyStore=DemoTrust"
   ```

   If you receive a host name verification error, and you cannot specify the host name in the URL to match the certificate, then you can add `-Dweblogic.security.SSL.ignoreHostnameVerification=true` to the `WLST_PROPERTIES` environment variable. This will bypass the host name verification.

3. Connect to the domain using the WLST as described in Invoking WLST in *Understanding the WebLogic Scripting Tool*.

> **Note:**
>
> The default address of an Administration Server in secured production mode is `t3s://`*`hostname`*`:9002`. Note the `s` in `t3s` and the port number.

Starting Managed Servers using Demo Keystores using a Start Script

If your domain is using demo keystores, you can start Managed Servers using a start script.

1. Create a boot identity file for each Managed Server. See Creating Boot Identity Files for Managed Servers in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

   Make sure to set appropriate permissions on the file and its containing folder. We recommend setting `chmod 600` on the `boot.properties` file and `chmod 740` for the folder, *`DOMAIN_HOME`*`/servers/`*`managedServerName`*`/security/`. If a `boot.properties` file is not available, then you must include a username and password when starting servers from the command line.

2. Run the `startManagedWebLogic` script as described in Starting Managed Servers with a Startup Script in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

   For example:

   ```
   startManagedWebLogic.sh managedServerName https://adminHostname:adminPort
   ```

Stopping Servers with Demo Keystores

- To shut down a server instance from WebLogic Remote Console, see Stop a Server in *Oracle WebLogic Remote Console Online Help*.
- To shut down a server instance with a script, see Shutting Down Servers with a Stop Script in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

> **Note:**
>
> – If you have not added arguments for your keystores to the `JAVA_OPTIONS` environment variable, then you must include them when stopping the server.
>
> – If you do not have a `boot.properties` file configured, then you will need to include the username and password at the command line.
>
> – If you want to use HTTPS instead of T3s as the protocol in the domain URL , you must enable tunnelling and default internal servlets. Set `ServerMBean.TunnelingEnabled` to `true` and `ServerMBean.DefaultInternalServletsDisabled` to `false`.

For example:

```
export JAVA_OPTIONS="-Dweblogic.security.TrustKeyStore=DemoTrust"
stopManagedWebLogic.sh managedServerName t3s://adminHostname:adminPort
[wlsUsername wlsPassword]
# To stop the Administration Server
export ADMIN_URL="t3s://adminHostname:adminPort"
stopWebLogic.sh [wlsUsername wlsPassword]
```

Using Secured Production Mode with Demonstration Keystores with KSS

You may need to perform additional configuration to support the use of the insecure demo keystores provided through the Keystore Service in domains with secured production mode enabled.

The Oracle Platform Security Service (OPSS) Keystore Service (KSS) provides a central management and storage of keys and certificates for all servers in a domain. See Configuring Oracle OPSS Keystore Service.

The OPSS KSS is only available with the JRF template and is not available to the default WebLogic Server configuration. Some of the following settings are already configured by the JRF template by default. If you have not modified those default values, you may be able to skip certain steps, such as configuring SSL/TLS or explicitly enabling KSS.

1. Set the domain mode to secured production mode using one of the following methods:
   - Create a new domain and select secured production mode as the domain mode when prompted.

     Creating a new domain allows you to configure many of the settings that are necessary for secured production mode in a single procedure and reduces the likelihood of conflicting configurations.

     See Creating a WebLogic Domain in *Creating WebLogic Domains Using the Configuration Wizard*.
   - Modify an existing domain to use secured production mode.

     See Changing the Domain Mode.
2. Choose one of the options for starting an Administration Server as described in Starting and Stopping Servers in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

   You may want to create a boot identity file to store user credentials for starting and stopping an instance of WebLogic Server. See Creating a Boot Identity File for an Administration Server in *Administering Server Startup and Shutdown for Oracle WebLogic Server*. If you choose to create a boot identity file, then make sure to set appropriate permissions on the `boot.properties` file and its containing folder, *`DOMAIN_HOME`*`/servers/AdminServer/security/`. We recommend setting `chmod 600` on `boot.properties` and `chmod 740` for `DOMAIN_HOME/servers/AdminServer/security`.
3. Connect to the domain using WebLogic Remote Console as described in Connect to an Administration Server in *Oracle WebLogic Remote Console Online Help*.

   The default address of your Administration Server is now `https://`*`hostname`*`:9002` . Note the `s` in *https* and the port number.

   In secured production mode, the non-SSL/TLS listen port (typically 7001) is disabled by default and traffic is routed over the SSL/TLS ports. If you want to configure listen ports, see Specify Listen Ports in *Oracle WebLogic Remote Console Online Help*.
4. Make sure WebLogic Server is using the demo keystores provided by KSS.
   a. In WebLogic Remote Console, in the **Edit Tree**, go to **Environment**, then **Domain**.
   b. On the **Security** tab, click **Show Advanced Fields**.
   c. Turn on the **Use KSS for Demo** option.
   d. Click **Save**.

5. Configure SSL/TLS as described in Set Up TLS in *Oracle WebLogic Remote Console Online Help*.
6. Enable host name verification, as described in Enable Host Name Verification in *Oracle WebLogic Remote Console Online Help*.
7. Save and commit your changes and then restart the Administration Server.
8. If your domain includes Managed Servers, and you plan to use WebLogic Remote Console to start them, perform these steps too:
   a. Ensure Node Manager is configured properly to work with WebLogic Remote Console, as described in Start Managed Servers in *Oracle WebLogic Remote Console Online Help*.
   b. Add `UseKSSForDemo=true` to the `nodemanager.properties` file.

      See Node Manager Properties in *Administering Node Manager for Oracle WebLogic Server*.
   c. Add startup arguments for your keystores for the Managed Server, as described in Configure Startup Arguments for a Managed Server in *Oracle WebLogic Remote Console Online Help*.

      Add `-Dweblogic.ssl.AcceptKSSDemoCertsEnabled=true`
   d. If multiple server instances run on the same computer in the domain and the domain-wide administration port is enabled, then you must perform one of the following:
      - Host the server instances on a multi-homed machine and assign each server instance a unique listen address
      - Override the domain-wide port on all but one of the servers instances on the machine. On the **Environment**: **Servers**: *myServer* page for each managed server, enter a unique port value in the **Local Administration Port Override** field.
   e. Save and commit your changes.
   f. Start the Managed Server.

Using WLST on Domains using Demo Keystores with KSS

When using WLST, you must perform additional configuration to support the use of the insecure demo keystores in domains with secured production mode enabled.

1. Start the Administration Server.

   You may want to create a boot identity file to store user credentials for starting and stopping an instance of WebLogic Server. See Creating a Boot Identity File for an Administration Server in *Administering Server Startup and Shutdown for Oracle WebLogic Server*. If you choose to create a boot identity file, then make sure to set appropriate permissions on the `boot.properties` file and its containing folder, *`DOMAIN_HOME`*`/servers/AdminServer/security/`. We recommend setting `chmod 600` on `boot.properties` and `chmod 740` for `DOMAIN_HOME/servers/AdminServer/security`.
2. Update the `WLST_PROPERTIES` environment variable to configure keystores:

   ```
   export WLST_PROPERTIES="-Dweblogic.security.SSL.trustedCAKeyStore=WL_HOME/
   server/lib/DemoTrust.jks -Dweblogic.ssl.AcceptKSSDemoCertsEnabled=true"
   ```

3. Connect to the domain using the WLST as described in Invoking WLST in *Understanding the WebLogic Scripting Tool*.

> **Note:**
>
> The default address of an Administration Server in secured production mode is `t3s://hostname:9002`. Note the `s` in `t3s` and the port number.

Starting Managed Servers using Demo Keystores with KSS using a Start Script

If your domain is using demo keystores with KSS, you can start Managed Servers using a start script.

1. Create a boot identity file for each Managed Server. See Creating Boot Identity Files for Managed Servers in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

   Make sure to set appropriate permissions on the file and its containing folder. We recommend setting `chmod 600` on the `boot.properties` file and `chmod 740` for the folder, `DOMAIN_HOME/servers/managedServerName/security/`. If a `boot.properties` file is not available, then you must include a username and password when starting servers from the command line.

2. Run the `startManagedWebLogic` script, adding `-Dweblogic.ssl.AcceptKSSDemoCertsEnabled=true`. The Managed Server start script is described in Starting Managed Servers with a Startup Script in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

   For example:

   ```
   startManagedWebLogic.sh managedServerName https://adminHostname:adminPort -
   Dweblogic.ssl.AcceptKSSDemoCertsEnabled=true
   ```

Stopping Servers with Demo Keystores with KSS

- To shut down a server instance from WebLogic Remote Console, see Stop a Server in *Oracle WebLogic Remote Console Online Help*.
- To shut down a server instance with a script, see Shutting Down Servers with a Stop Script in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

> **Note:**
>
> – If you have not added arguments for your keystores to the `JAVA_OPTIONS` environment variable, then you must include them when stopping the server.
>
> – If you do not have a `boot.properties` file configured, then you will need to include the username and password at the command line.
>
> – If you want to use HTTPS instead of T3s as the protocol in the domain URL , you must enable tunnelling and default internal servlets. Set `ServerMBean.TunnelingEnabled` to `true` and `ServerMBean.DefaultInternalServletsDisabled` to `false`.

For example:

```
export JAVA_OPTIONS="-
Dweblogic.security.SSL.trustedCAKeyStore=WL_HOME/server/lib/DemoTrust.jks -
Dweblogic.ssl.AcceptKSSDemoCertsEnabled=true"
# To stop a Managed Server
./stopManagedWebLogic.sh managedServerName t3s://adminHostname:adminPort
[wlsUsername
wlsPassword]
# To stop the Administration Server
export ADMIN_URL="t3s://adminHostname:adminPort"
./stopWebLogic.sh [wlsUsername wlsPassword]
```

Using Secured Production Mode without SSL/TLS

By default, when domains in secured production mode start, they use the default SSL/TLS and Administration Channel. If your domain or its start mechanism is not configured properly, you will not be able to connect. However, you can modify the domain to disregard the SSL/TLS requirements.

1. In WebLogic Remote Console, in the **Edit Tree**, go to **Environment**, then **Domain**.
2. Turn on the **Listen Port Enabled** option.
3. Turn off the **Enable Administration Port** option.
4. Turn off the **SSL Enabled** option.
5. Click **Save**.
6. If your domain contains clusters, then go to **Environment**, then **Clusters**. On each cluster, make the following change:
   a. On the **Replication** tab, turn off **Secure Replication Enabled**.
   b. Click **Save**.
7. Commit your changes.
8. Restart your Administration Server and all of your Managed Servers.

You can also use WLST Offline to disable the SSL/TLS requirements. See Example: Disabling TLS/SSL on a Domain in Secured Production Mode in *Understanding the WebLogic Scripting Tool*.

Part VIII
Appendixes

Supplemental and reference information for Oracle WebLogic Server security.

A

Keytool Command Summary

The keytool commands are commonly used for creating and using JKS and PKCS12 keystores with Oracle WebLogic Server.
In Table A-1, an option surrounded by brackets ([]) indicates that if you omit the option from the command, you are subsequently prompted to enter that option's value. For example, if you follow Oracle's *strong* recommendation to omit command options for specifying passwords, you are prompted for those passwords after you enter the command, as in the following example. (User input is shown in **bold**.)

```
C:\DOMAIN_NAME>keytool -genkeypair -keystore MyKeyStore
Enter keystore password:
Re-enter new password:
```

Unlike passwords that are specified in command-line options, a password entered in response to a prompt is not echoed in the command window and is not captured in logs. This practice helps keep your passwords secure.

For detailed documentation for the Java keytool utility, see the `keytool` utility section in JDK Tool Specifications.

Table A-1 Commonly Used keytool Commands

Command	Description
`keytool -genkeypair -keystore keystorename -storepass keystorepassword -storetype keystoretype`	Generates a key pair (a public key and associated private key) and self-signed digital certificate in a keystore. If the keystore does not exist, it is created.
`keytool -importcert -alias aliasforprivatekey -file privatekeyfilename.pem -keyfilepass privatekeypassword -keystore keystorename -storepass keystorepassword -storetype keystoretype`	Updates the self-signed digital certificate with one signed by a trusted CA.

`keytool -importcert -alias rootCA -trustcacerts -file RootCA.pem -keystore trust.jks -storepass keystorepassword -storetype keystoretype`
 	
`keytool -importcert -alias intermediate -trustcacerts -file Intermediate.pem -keystore keystorename -storepass keystorepassword -storetype keystoretype`
	Creates a custom keystore to be used for holding an intermediate CA certificate.
	• The first keytool command creates the keystore, `trust.jks`, which holds the root CA certificate.
	• The second keytool command imports the intermediate CA certificate into `trust.jks`.
	This enables WebLogic Server's SSL implementation to transmit the intermediate certificate with the server's public certificate to the client during the SSL handshake.

Table A-1 (Cont.) Commonly Used keytool Commands

Command	Description
`keytool -importcert -alias aliasfortrustedca -trustcacerts -file trustedcafilename.pem -keystore keystorename -storepass keystorepassword -storetype keystoretype`	Loads a trusted CA certificate into a keystore. If the keystore does not exist, it is created.
`keytool -certreq -alias alias -sigalg sigalg -file certreq_file -keyfilepass privatekeypassword -storetype keystoretype -keystore keystorename -storepass keystorepassword`	Generates a Certificate Signing Request (CSR), using the PKCS#10 format, and a self-signed certificate with a private key. Stores the CSR in the specified *certreq_file*, and the certificate/private key pair as a key entry in the specified keystore under the specified alias.
`keytool -list -keystore keystorename`	Displays the contents of the keystore.
`keytool -delete -keystore keystorename -storepass keystorepassword -alias privatekeyalias`	Deletes the entry identified by the specified alias from the keystore.
`keytool -help`	Provides online help for keytool.

B

Interoperating With Keystores From Prior Versions

Learn how to use keystores in WebLogic Server version 14.1.2.0.0 or later with keystores in a previous release of Oracle WebLogic Server.
If you are using WebLogic Server 14.1.2.0.0 or later together with an earlier version of WebLogic Server, be aware that the behavior of the demo CA certificate and demo certificates changed in 14.1.2.0.0.

From WebLogic Server 12.1.2 to 14.1.1.0.0, all installations of WebLogic Server shared the same demo CA and its paired private key. As of WebLogic Server 14.1.2.0.0, a unique demo CA is generated for each domain.

WebLogic Server 12.2.1.4.0 through 14.1.1.0.0

If you *upgrade* to WebLogic Server 14.1.2.0.0 or later from a previous release, then your upgraded domain will continue to use the demo CA certificate and demo certificates from its previous release.

If you plan to use WebLogic Server 14.1.2.0.0 or later together with a previous release, then be aware of the following changes:

- Whenever a new domain is created, it generates a unique demo CA certificate.
- The demo identity keystore is now in PKCS12 format and located at *DOMAIN_HOME*`/security/DemoIdentity.p12`
- The demo trust keystore is now in PKCS12 format and located at *DOMAIN_HOME*`/security/DemoTrust.p12`
- The expiration period for the new demo CA certificate and demo certificates is significantly shortened compared to previous releases. The demo certificates expire after 6 months and the demo CA certificate expires after 5 years. For continuity of service, renew them prior to expiration.